[Salling 1998] L. Salling, *Formella språk, automater och beräkningar*, 1998, ISBN 91-630-7707-8

[Schmeiser and Barnard 1995] J. Schmeiser, D. Barnard, "Producing a top-down parse order with bottom-up parsing", *Information Processing Letters vol. 54/6*, Elsevier, 1995, pp. 323-326

[Suárez et al. 1999] A. Suárez, M. Saiz-Noeda and M. Palomar, "A method of restricted knowledge acquisition from WordNet", *Knowledge-Based Intelligent Information Engineering Systems*, IEEE, 1999, pp. 38-41

[Taylor et al. 2001] A. Taylor, M. Marcus and B. Santorini, "The Penn Treebank - An Overview", 2001, http://treebank.linguist.jussieu.fr/pdf/1.pdf, last viewed 2004-05-04

[Valenti 2003] S. Valenti, F. Neri and A. Cucchiarelli, "An Overview of Current Research on Automated Essay Grading", http://jite.org/documents/Vol2/v2p319-330-30.pdf, *Journal of Information Technology Education vol. 2*, Journal of Information Technology Education, 2003, pp. 319-330

[WWW CDT] Center for Democracy & Technology, "Why Am I Getting All This Spam?", Unsolicited Commercial E-mail Research Six Month Report, 2003, http://www.cdt.org/speech/spam/030319spamreport.shtml, last viewed 2004-03-11

[WWW CL1] W3-Corpora project, "Introduction: Corpus Linguistics", http://www.essex.ac.uk/linguistics/clmt/w3c/corpus_ling/content/introduction2.html, last viewed 2004-05-04

[WWW CL2] T. McEnery, A. Wilson, "Corpus Linguistics: Part Two: What is a Corpus, and what is in it?", http://www.ling.lancs.ac.uk/monkey/ihe/linguistics/corpus2/2fra1.htm, last viewed 2004-05-04

[WWW GoE] The Survey of English Usage, "The Internet Grammar of English", http://www.ucl.ac.uk/internet-grammar/home.htm, last viewed 2004-05-04

[WWW LCA] D. Arnold, "Bottom-up parsing: Left-corner algorithm", http://courses.essex.ac.uk/lg/LG511/7-LeftC/index.pdf, 2002, last viewed 2004-05-04

[WWW MT] "Machine Translation - Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/Machine_translation, 2004, last viewed 2004-06-09

[WWW NLPTP] P. Blackburn, K. Striegnitz, "Natural Language Processing Techniques in Prolog", http://www.coli.uni-sb.de/~kris/nlp-with-prolog/html/index.html, last viewed 2004-05-04

[Liu 2003] F. Liu, L. Gu, Y. Gao and M. Picheny, “Use of Statistical N-Gram Models in Natural Language Generation for Machine Translation”, *Acoustics, Speech, and Signal Processing*, IEEE, 2003, pp. 636-639

[van Lohuizen 1998] M.P. van Lohuizen, ”Architectural Aspects of Natural Language Processing Systems”, PDS technical report PDS-1998-001, Delft University of Technology, 1998

[Loper 2004] Edward Loper, “NLTK: Building a pedagogical toolkit in Python”, PyCon DC 2004, http://www.python.org/pycon/dc2004/papers/35/nltk.pdf, last viewed 2004-05-04

[Knott et al. 2002] A. Knott, I. Bayard, S. de Jager and N Wright, “An architecture for bilingual and bidirectional NLP”, *Proceedings of the 2nd Australasian Natural Language Processing Workshop*, 2002, pp. 63-70

[Krotov et al. 1999] A. Krotov, M. Hepple, R. Gaizauskas, Y. Wilks, “Compacting the Penn Treebank Grammar”, arXiv, 1999

[Marcus et al. 1999] M. P. Marcus, B. Santorini and M. A. Marcinkiewicz, “Building a large annotated corpus of English: the Penn Treebank”, *Computational Linguistics*, vol. 19, 1993, pp. 313-330

[McKelvie 1997] D. McKelvie, C. Brew and H. Thomson, “Using SGML as a basis for data-intensive NLP”, *Proceedings of the fifth conference on Applied natural language processing*, Morgan Kaufmann Publishers Inc, 1997, pp. 229-236

[Nederhof 1993] M. Nederhof, “Generalized left-corner parsing”, *European Chapter Meeting of the ACL*, Association for Computational Linguistics, http://acl.ldc.upenn.edu/E/E93/E93-1036.pdf, 1993, pp. 305-314

[Niemann 2001] T. Niemann, “A compact guide to Lex & Yacc”, http://epaperpress.com/lexandyacc/, 2001, last viewed 2004-05-04

[Nii 1986] H. Nii, “The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures”, *AI Magazine vol. 7/2*, 1986, pp. 38-53

[Nirenburg 1995] S. Nirenburg, “The Pangloss Mark III Machine Translation System”, ftp://ftp.cs.cmu.edu/afs/cs.cmu.edu/project/cmt-16/pangloss/www/papers/pangloss.ps, 1995, last viewed 2004-05-04

[Palmer 2002] J. Palmer, R. Williams, H. Dreher, “Automated Essay Grading System Applied to a First Year University Subject - How Can We do it Better?”, http://ecommerce.lebow.drexel.edu/eli/2002Proceedings/papers/Palme026Autom.pdf, Informing Science, 2002, pp. 1221-1229

[Pereira 2000] F. Pereira “Formal grammar and information theory: Together again?”, *Philosophical Transactions of the Royal Society*, 358(1769):1239-1253, April 2000

[Drake 2003] M. A. Drake, *Encyclopedia of Library and Information Science,* Marcel Dekker, 2003, ISBN 0-8247-4259-1, pp. 2126-2136

[Erman et al. 1980] L. Erman, F. Hayes-Roth, V. Lesser, D. Reddy, “The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty”, *ACM Computing Surveys vol. 2/12*, ACM Press, 1980, pp. 213-253

[Grishman 1995] R. Grishman, “TIPSTER Phase II Architecture Design Document (Tinman Architecture) Version 1.52”, http://www.cs.nyu.edu/cs/faculty/grishman/tipster.html, last viewed 2004-05-04

[Grosz 1986] B. J. Grosz, K.S. Jones, B.L. Webber, *Natural Language Processing*, Morgan Kaufmann Publishers Inc, 1986, ISBN 0-934613-11-7

[van Halteren et al. 1998] H. van Halteren, J. Zavrel and W. Daelemans, ”Improving data-driven wordclass tagging by system combination”, In Proc. COLING/ACL98, 1998, pp. 491-497, Montreal, Canada.

[Hearst 2000] M.A. Hearst, ”The debate on automated essay grading”, *IEEE Intelligent Systems*, IEE/IEEE, 2000, pp. 22-37

[Hemming and Lassi 2002] C. Hemming, M. Lassi, “Copyright and the Web as Corpus”, 2002, http://www.adm.hb.se/personal/mol/gslt/copyright.pdf, last viewed 2004-05-04

[Hewett and Hewett 1993] M. Hewett, R. Hewett, “A language and architecture for efficient blackboard system”, *Ninth Conference on Artificial Intelligence for Applications, Proceedings*, IEEE Comput. Soc. Press, 1993, pp. 34-40

[Jurafsky and Martin 2000] D. Jurafsky, J. H. Martin, *Speech and Language Processing*, Prentice Hall, 2000, ISBN 0-13-095069-6

[Karkaletsis 1999] V. Karkaletsis, G. Paliouras, G. Petasis, N. Manousopoulou, C. Spyropoulos, “Named-Entity Recognition from Greek and English Text”, *Journal of Intelligent and Robotic Systems*, Kluwer, 1999, pp. 123-135

[Kruengkrai 2003] C. Kruengkrai, C. Jaruskulchai, “Generic Text Summarization Using Local and Global Properties of Sentences”, *Web Intelligence*, IEEE, 2003, pp. 201-206

[Leidner 2003] J. L. Leidner, “Current Issues in Software Engineering for Natural Language Processing”, *Proceedings of the Workshop on Software Engineering and Architecture of Language Technology Systems (SEALTS)*, 2003, pp. 45-50

[Liere et al. 1998] R. Liere, J. Harkes, W. Leeuw, “A distributed blackboard architecture for interactive data visualization”, *Visualization '98. Proceedings*, IEEE, 1998, pp. 225-231

[Lillian 2003] L. Lillian, “'I'm sorry Dave, I'm afraid I can't do that': Linguistics, Statistics, and Natural Language Processing circa 2001”, arXiv, 2003

REFERENCES

[Allen 1987] J. Allen, *Natural Language Understanding*, The Benjamin/Cummings Publishing Company, 1987, ISBN 0-8053-0330-8

[Bates and Weischedel 1993] M. Bates and R. Weischedel, *Challenges in Natural Language Processing*, Cambridge University Press, 1993, ISBN 0521410150

[Boitet and Seligman 1994] C. Boitet, M. Seligman, "The 'Whiteboard' Architecture: a way to integrate heterogeneous components of NLP systems", *Proceedings of COLING-94*, Vol. I, 1994, pp. 426-430

[Brill and Wu 1998] Eric Brill and Jun Wu, "Classifier combination for improved lexical disambiguation", Proceedings of COLING-ACL '98, Association for Computational Linguistics, 1998, pp. 191-195

[Choi 1999a] F. Y. Y. Choi, "A flexible distributed architecture for NLP system development and use", Proceedings of ACL'99 (student session), 1999, pp. 615-618

[Choi 1999b] F. Y. Y Choi, "An adaptive voting mechanism for improving the reliability of NLP systems", submitted to EACL'99 (student session), 1999, http://www.cs.man.ac.uk/~mary/choif/publication.html, last viewed 2004-05-04

[Chomsky 1957] N. Chomsky, "Syntactic Structures", Mouton, The Hague, 1957

[Crnkovic 2003] I. Crnkovic, "Component-based software engineering - new challenges in software development", Proceedings of the 25th International Conference on Information Technology Interfaces 2003, IEEE, 2003, pp. 9-18

[Cunningham 1997] H. Cunningham, "Information Extraction – A User Guide", arXiv, 1997

[Cunningham et al. 1997] H. Cunningham, K. Humphreys, R. Gaizauskas and Y. Wilks, "Software Infrastructure for Natural Language Processing", arXiv, 1997

[Cunningham 2000] H. Cunningham, "Software Architecture for Language Engineering", Ph.D. thesis, University of Sheffield, 2000, http://gate.ac.uk/sale/thesis/, last viewed 2004-05-04

[Cunningham et al. 2002a] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, "GATE: an Architecture for Development of Robust HLT Applications", Proceedings of the 40th Annual Meeting of the ACL, Philadelphia, July 2002, pp. 168-175

[Cunningham et al. 2002b] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, and C. Ursu, "The GATE User Guide", 2002, http://gate.ac.uk/, last viewed 2004-05-04

17. In the treeview, choose “GATE → Applications” and double click on the pipeline under “Applications”.

18. Add the processing resources in the same order as in 4, by double clicking each processing resource.

19. For each processing resource, select it and add the document chosen in 2, by selecting it in the document parameter combo box.

20. Click on SenseTaggerPR and choose level of sensitivity and write the path to “sense_information.txt” (for example, “C:\Program\SenseTaggerPR\classes\sense_information.txt”).

21. Click on SemanticAnalyzerPR and write the path to “file_properties.xml” (for example “C:\Program\JWNL13rc3\file_properties.xml”).

22. Click on “Run”.

Semantic Analyzer

Before starting GATE, these steps have to be made:

1. Download and install WordNet 2.0[21]

2. Download and extract the WordNet interface JWNL 1.3rc3[22] to a directory named “jwnl13rc3”.

3. Remove the package “net” located in “Gate 2.2\bin\Gate.jar”.

4. Copy the package “net” and the files “JWNLResource_en.properties” and “PrincetonResource_en.properties” located in “jwnl.jar” in “jwnl13rc3” to “Gate 2.2\bin\Gate.jar”.

5. Copy the package “org” located in “commons-logging.jar” in “jwnl13rc3” to “Gate 2.2\bin\Gate.jar”.

6. Edit “file_properties.xml” in “jwnl13rc3” by replacing the value on the line with the path to WordNet’s dictionary to where WordNet is really installed. The line looks like this: <param name="dictionary_path" value="[The path to the WordNet dictionary]"/>

Follow these simple steps to get the Semantic Analyzer to work:

12. Start GATE.

13. Choose “File → GATE Document”. Add any document, preferably containing an actor-receiver scenario, explained in Chapter 7.2 (for example “test.txt” in the same directory as SemanticAnalyzerPR).

14. Choose “File → Load a CREOLE repository”. Add the directories where these files are located:
    - SenseTaggerPR.jar and creole.xml
    - SemanticAnalyzerPR.jar and creole.xml

15. Choose “File → New processing resource”. Add these processing resources:
    - Document Reset PR
    - ANNIE English Tokeniser
    - ANNIE English Sentence Splitter
    - SenseTaggerPR
    - SemanticAnalyzerPR

16. Choose “File → New Application → Pipeline”.

[21] http://www.cogsci.princeton.edu/~wn/
[22] http://sourceforge.net/projects/jwordnet

APPENDIX B - INSTALLATION INSTRUCTIONS

Email Extractor

Follow these simple steps to get the Email Extractor to work:

1. Start GATE.

2. Choose "File → GATE Document". Add any document, preferably containing one or more emails (for example test.txt in the same directory as the Email Extractor PR).

3. Choose "File → Load a CREOLE repository". Add the directory where these files are located:
   - Email Extractor JAR and creole.xml file.

4. Choose "File → New processing resource". Add these processing resources:
   - Document Reset PR
   - ANNIE English Tokeniser
   - ANNIE English Sentence Splitter
   - Email Extractor

5. Choose "File → New Application → Pipeline".

6. In the treeview, choose "GATE → Applications" and double click on the pipeline under "Applications".

7. Add the processing resources in the same order as in 4, by double clicking each processing resource.

8. For each processing resource, select it and add the document chosen in 2, by selecting it in the document parameter combo box.

9. Press "Run".

10. Double-click the document.

11. Click Annotation Sets and select "Email-Address". All emails will be marked with a color.

Visual Resource

XML
Extensible Markup Language

Yacc
Yet another compiler compiler

Lex
Lexical analyzer

LR
Language Resource

LT-NSL
Not found, probably: Language Technology Group Normalised SGML Library

LT-XML
Not found, probably: Language Technology Group Extensible Markup Language

MASTOR
Multilingual Automatic Speech-to-Speech Translator

MT
Machine Translation

MULTEXT
Multilingual Text Tools and Corpora

NLP
Natural Language Processing

NLTK
Natural Language Toolkit

NLU
Natural Language Understanding

PR
Processing Resource

POS
Part-of-speech

SGML
Standard Generalized Markup Language

TE
Text Engineering

TEA
Text Engineering Architecture

TIPSTER
Not an acronym; the name of a US IE/IR research programme

VR

APPENDIX A - ACRONYMS

There are numerous acronyms in this thesis which might be hard to memorize. Therefore this list has been compiled.

ALEP
Advanced Language Engineering Platform

ANNIE
A Nearly New Information Extraction system

ATIS
Air Traffic Information System

BNF
Backus-Naur Form

CFG
Context-free Grammar

CREOLE
Collection of Reusable Objects for Language Engineering

DARPA
The Defense Advanced Research Projects Agency

DBMS
Database Management System

ENGTWOL
Not found

EPSRC
U.K. Engineering and Physical Sciences Research Council

GATE
General Architecture for Text Engineering

HMM
Hidden Markov Model

IE
Information Extraction

IR
Information Retrieval

JWNL
Java WordNet Library

CONCLUSION AND FUTURE WORK

This thesis has explored the subject of natural language processing from a software engineering perspective. We have explained fundamental concepts (corpora, lexicon and grammar) and fundamental algorithms (POS tagger and grammar parser). We have also discussed serial and parallel architectures and shown in our experiments how an NLP system can be built using these building-blocks in a serial architecture. We did not come in contact with any other type of architecture during our empirical evaluation. However, we came to the conclusion that parallel architectures are suitable when there is more than one way to solve a problem and when those results can be merged for improved results.

The suitable frameworks and toolkits we found were GATE and NLTK, of which only GATE was experimented with because we found it even more suitable to develop complete NLP systems. The empirical evaluation gave a practical view of the topics covered in our literature study and was in our opinion a successful part of the thesis. We also believe that these types of initiatives, in combination with a highly portable language, will contribute to increase the reusability in the NLP community, which currently seems to be low.

The recent emergence of NLP related applications and development environments combined with the increase of computer performance (which we believe will lead to the release of even more NLP applications) reveals a bright future for the NLP community.

Our work leaves many trails to be followed. Possible future work could be to delve deeper into different NLP algorithms, or maybe to research about a specific NLP subject such as speech recognition. It would also be interesting to evaluate another framework or toolkit such as NLTK.

spectra of NLP developments aids. Taking into account that we evaluated all that we found and chose what we considered to be the most suitable for our purpose we believe that we can still draw some conclusions about the systems. Considering the small number of systems there are, and the quality of GATE, we can say that it is a step in the right direction, but a long way before reaching what we call mature. Although we cannot say GATE was a dream to work with, we still do recommend it. After all, we did get our work done despite the experienced quirks.

We were very impressed with the power of WordNet. Using the feature "Derivationally related forms" we facilitated basic semantic analyzing. However, we are still disappointed with the lack of support for that feature in JWNL.

Having researched quite a lot, we found an increase in NLP related applications and projects aimed towards development of NLP applications during the last years. We suspect one of the reasons for this is the increase in computer performance, which allow for complex NLP tasks such as speech recognition. Our conclusion leads us to believe that there will be an even greater increase in such applications in the future, as the performance of computers is steadily rising, revealing a bright future for NLP.

We feel that the purpose of this thesis has been met. Our literature study has been quite thorough covering important fundamental aspects of NLP. Combining this with experiments to give a more practical view of the topic was in our opinion successful.

8.3 What we are satisfied with

We are very satisfied with our thorough literature study. Also, by experimenting we could connect the theory part to our empirical evaluation which proved enlightening. During the evaluation, we directly felt very familiar with the concepts in GATE (such as corpus, POS tagger, and so on). This made us feel that we had written about concepts relevant to the field of NLP.

8.4 What we could have done better

We spent a lot of time on the literature study, and it is possible that we could have given more time to the empirical evaluation and covered more concepts, or even tried out NLTK. There is also a possibility that it would have been more appropriate to cover other parts in the experiments than what we did, but it is hard to tell even afterwards. In any case it was quite hard to decide which experiments to conduct.

If we had more time at our disposal, we could have further validated the results by, for example, doing a questionnaire, asking NLP researches what they believe is the answers to our research questions.

Another questionable decision was to choose experiments based on the already chosen framework. However, this allowed us to spend our time more effectively since we did not waste time with experiements which would be too difficult, or even impossible, to implement.

Reliability

When developing the processing resources in GATE we did not really experience any reliability problems. Although our programs were quite small, we do believe that Java helps reducing errors. Especially, it avoids memory leaks due to the garbage collector, and problems related to pointers because such low level concepts are not included in the language.

8.2 The results

The field of NLP is quite large, and although we have delved deep into key fundamental concepts, there is much more yet to be explored. Ironically, during our literature study the most difficult topics to find information about were the most basic ones. We believe the reason for this is because most articles we have read are intended for people already familiar with NLP concepts, so they take for granted that fundamental concepts do not have to be explained.

We found three different fundamental concepts, namely corpora, lexicon and grammar. We have covered these quite thorough, and given numerous real life excerpts. When it comes to the fundamental algorithms we have covered how a tagger works, and also how to parse sentences using a grammar, which both indeed are very central in the development in NLP applications. In our practical work, we did not come in contact with any of the fundamental algorithms, but we did come in contact with both corpus and lexicon. This is definitely something which could be improved upon, and something that could be considered future work. We simply did not have enough time.

Having conducted the literature study we have learnt that you do not necessarily have to be a linguist to create an NLP application. One indication of that is that the experiments we conducted were successful even though we are far from being linguists. Another indication is the fact that you can create, for example, a POS tagger using statistical knowledge and no deeper knowledge about language. However, it is important to see that in order to advance NLP and integrate more advance concepts such as actual understanding (i.e. getting NLP closer to its goal - NLU) linguistic knowledge is required.

We found both serial and parallel NLP architectures. The parallel architectures we found (aside from a simple parallel architecture) were blackboard and whiteboard. However, not a single framework or toolkit we came in contact with used parallel architectures. The result does not surprise us since almost all computers today only have one processor. Furthermore, in our experiments each module is dependent of another, which makes serial architectures logical. However, the Pangloss Mark III Machine Translation System is one example where a parallel architecture is advantageous, where three independent modules run in parallel and the results are combined to produce a superior result to any module in isolation. Similarly, we have seen that voting can be used where the results of different modules with the same goal but different ways to achieve them can be combined for improved accuracy.

The suitable frameworks and toolkits we found were GATE and NLTK, and after weighing their pros and cons, GATE was chosen to experiment with. Only working with one selected framework or toolkit can of course not be considered the whole

8 DISCUSSION

This section aims at discussing which software engineering related questions and problems can arise during the development of NLP applications. It also discusses the results we have achieved, or in other words the answers to the problem description. Finally, it discusses what we are satisfied with, and what we believe we could have done better.

8.1 Software engineering related questions and problems

As expected, during our empirical evaluation important software engineering related questions and problems related to the development of NLP systems arose:

Reusability

Both GATE and NLTK facilitate reusability by the use of components. The processing resources (PRs, or CREOLE components) we developed in GATE could easily be used by anyone else. We also used the built in ANNIE components in GATE and directly utilized the reusability aspects of GATE. J.L. Leidner [Leidner 2003, p. 2] argues that the global amount of reuse in NLP is low, especially on the software side. Although we are not in the position to argue with him, we believe that especially GATE has the potential to increase the reusability in this community. Because NLTK is primarily intended to be a pedagogical tool, we have a feeling that it will mostly attract students. We cannot prove this at all, but if it is true maybe it will not contribute in regard to the reusability in the NLP community as a whole.

Portability

Applications written in Java or Python are all portable without the need of recompilation. Python has interpretable code, while Java has Java Byte Code which is interpreted by a virtual machine. This means that the transition from one operating system to another should be unnoticeable. As we wrote before, GATE has switched its chosen language from C++ to Java, and we believe that portability is another important factor which advocated the change. Although C++ applications can be run on different platforms, they have to be compiled for each platform and also have to be programmed with portability in mind. We believe that portability promotes reusability, simply as it makes it easier when you can use components without recompilation across platforms.

Maintainability

GATE and NLTK promotes component based applications. Component based applications are easier to maintain and the reason for this is that an isolated faulty component can easily be replaced by a working one without affecting the rest of the system. Also, we believe that Java and Python are generally easier to read and understand than for example C++. We believe that the readability factor (and thus the maintainability factor) could be one of the reasons for GATE changing the language of choice from C++ to Java. Also, one of the goals of Python is that it should have a very clear syntax.

7.2.2 Known flaws

There are some flaws in the semantic analyzer, and they are:

- In the sentence “The stone called the plumber” we explained that if the sementic sensitivity is set to four, the sentence is probably not semantically correct. Probably, since in the context of a children’s story, it might very well be possible for a stone to call someone. Being able to analyze sentences with such complexity is not possible for any system today.

- The semantic sensitivity might not always be right. Different semantic sensitivities might be appropriate for different sentences.

- Sentences where the actor is a company or an office will be analyzed as probably being semantically incorrect. The reason for this is because a “caller” some sort of living object and a company or an office is not. However, this we more see as a flaw in WordNet.

- It is always the first sense of the actor which is analyzed. This means that with a semantic sensitivity of four, the sentence “The bear called the plumber” will be seen as semantically incorrect since the first sense of “bear” is that of a bear which has claws, even though it could be an investor.

7.2.3 Future work

We wanted the program to retrieve information about the derivationally related forms. Unfortunately, JWNL did not yet support that so we had to look it up for ourselves and write which words were associated with which actors in a text file (for example, “call” is associated to “caller”). When JWNL can retrieve derivationally related forms, the program can easily be modified to retrieve that information instead.

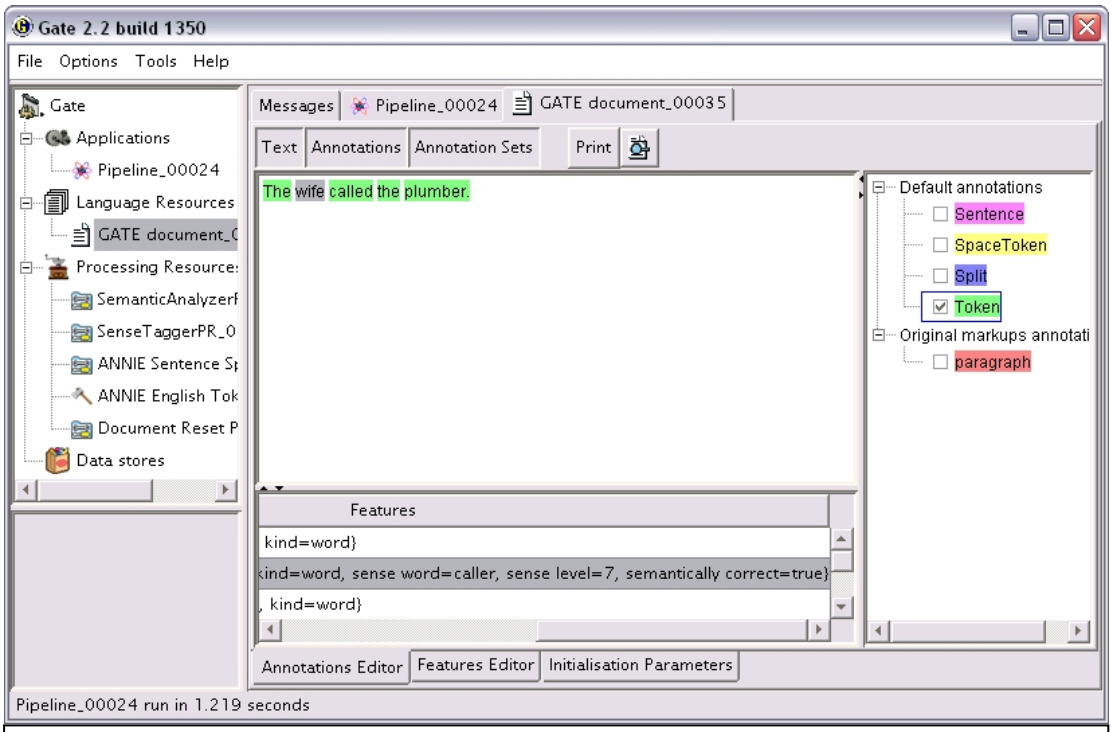

Figure 7.10. The semantic analyzer in action.

The figure show that ”wife” has been marked with ”caller” as its ”sense word”, that it is sense 7 of ”caller” that is referred to and the ”wife” probably is semantically correct in this sentence.

7.2.1 Problems encountered

A few problems arose during the development of the Semantic Analyzer. The problems were:

- **Conflicting JWNL versions:** We had much problem getting the WordNet interface Java WordNet Library (JWNL) to work. After some time we found out that GATE had a version of JWNL of its own, residing inside the file GATE.jar, causing some conflicts. GATE used an older version of JWNL so we simply replaced it in the GATE jar-file for it to work.

- **Unable to retrieve certain information:** When we tried to get the derivationally related forms of a word using JWNL we were unsuccessful. After some searching on the forums of JWNL, we found out that JWNL did not yet support the retrieval of that information.

The first problem took quite some time to find, which was very frustrating. On the other hand, since GATE itself used JWNL we believed that we had made a good choice when choosing a WordNet interface. The other problem did not cause a big delay.

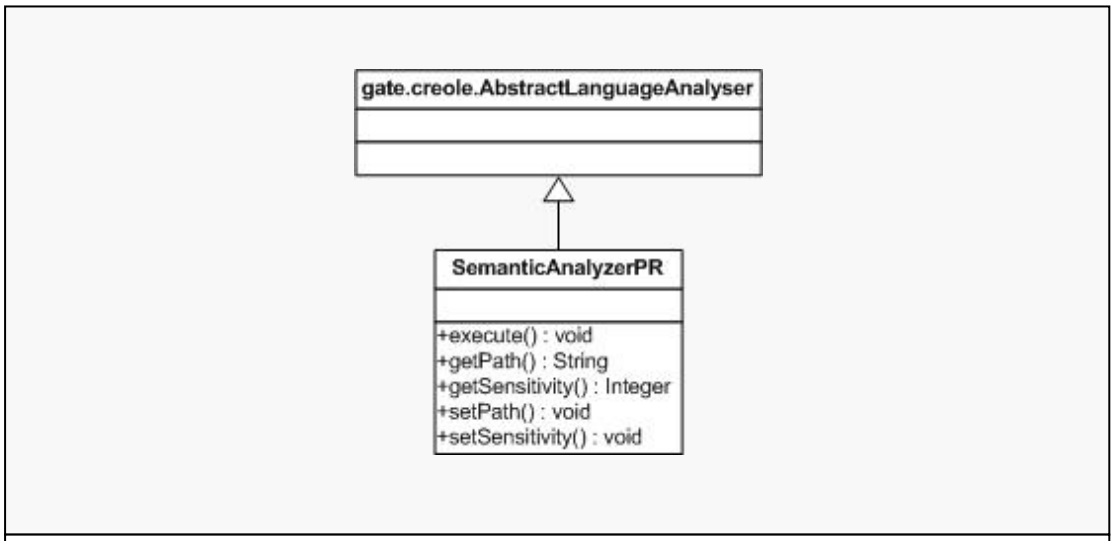

Figure 7.9. A UML diagram over SemanticAnalyzerPR.

The three classes that the system consists of are:

- **Sense:** This class stores information about a word's derivationally related form; the word (for example "called"), its sense word (for example "caller") and the sense level of the sense word (for example "7").

- **SenseTaggerPR:** The purpose of this processing resource is to tag the actor in an actor-reciever scenario with its derivationally related form (for example, tagging "wife" as a "caller" sense 7).

- **SemanticAnalyzerPR:** The purpose of this processing resource finds tags created by the SenseTaggerPR and then analyze if the sentence semantically correct.

Figure 7.10 show the semantic analyzer in action.

not semantically correct. On the other hand, if the word can be found, then there is a possibility that the sentence is semantically correct.

To clarify semantic sensitivity we will analyze sentence (1). With a semantic sensitivity of four, “person” is found in the “caller” tree. Then we search the “wife” tree to see if we can find “person”, which we do. If the sensitivity is three, the word found in the “caller” is “communicator”, which is not present in the “wife” tree. To further explain, take this sentence for example:

(2) The stone called the plumber

This sentence looks like (1) with the difference that it is a stone instead of a wife which called a plumber. With a semantic sensitivity of four, sentence (2) is not semantically correct, since “person” is not present in the “stone” tree.

After we had decided how the analyzing should be done (as explained above) we designed the system. Figure 7.8 and 7.9 show two UML diagrams illustrating the two processing resources that the Semantic Analyzer consists of.

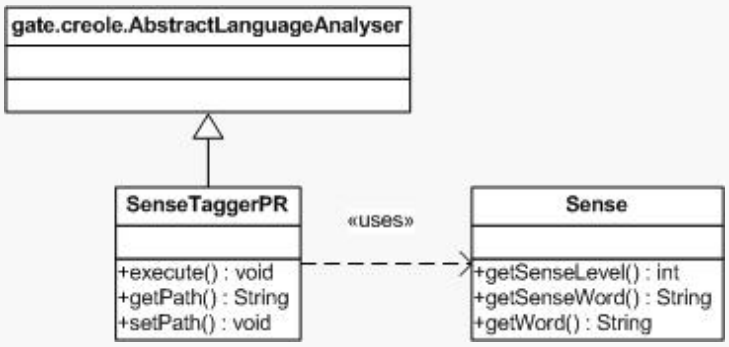

Figure 7.8. A UML diagram over SenseTaggerPR.

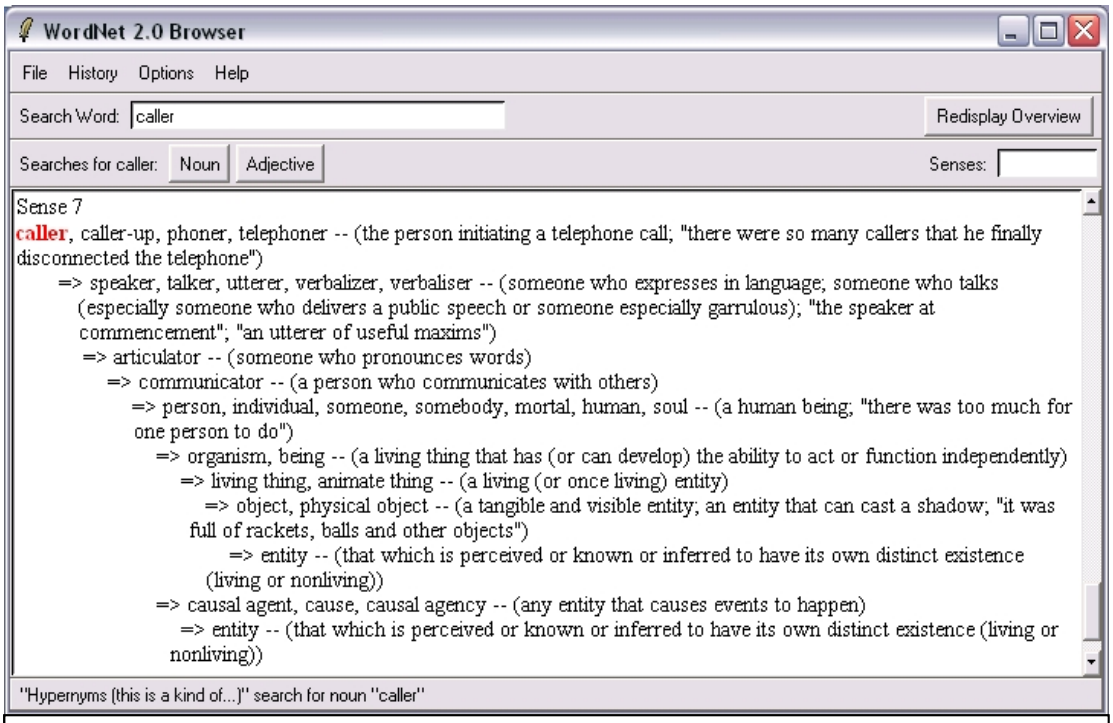

Figure 7.6. Hypernym tree of the noun “caller”, sense 7.

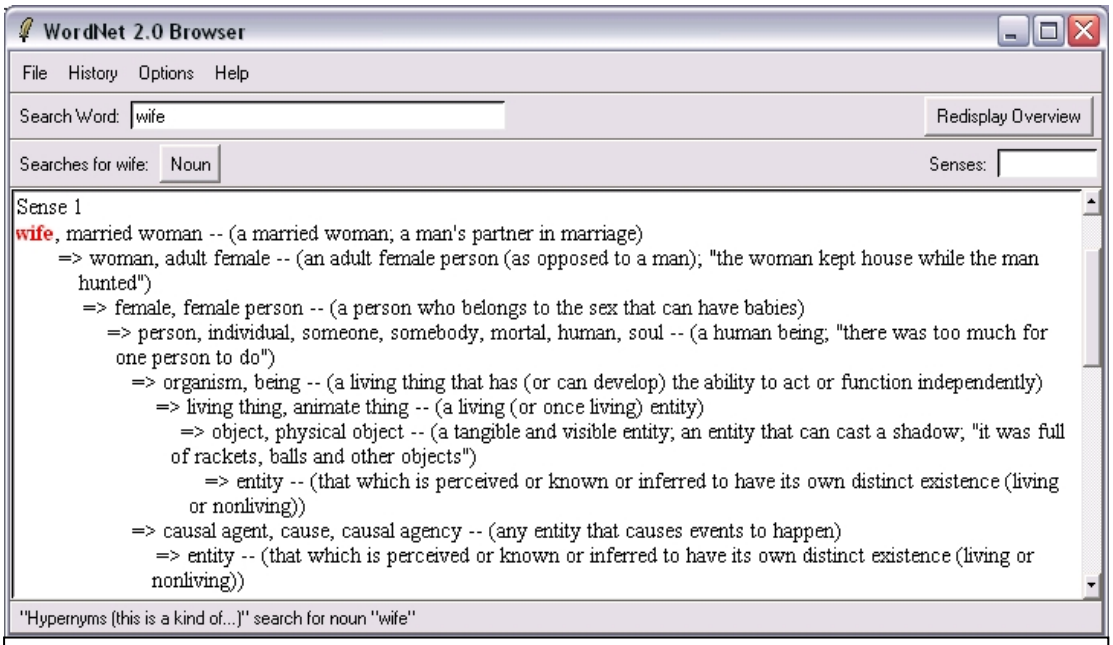

Figure 7.7. Hypernym tree of the noun “wife”, sense 1.

If we go down the “wife” tree we will eventually find a match on the “caller” tree. The first match is “person”. This is at the fourth level in the “caller” tree. However, we can not just look for a match between trees as there always will be a match between two nouns since they always end with them being an “entity”. Therefore we decided to introduce a concept of our own which we called *semantic sensitivity*. Semantic sensitivity is a number stating how long down we should search the hypernym tree of the derivationally related form. The word found is then compared to the actor tree. If the word cannot be found, there is a possibility that the sentence is

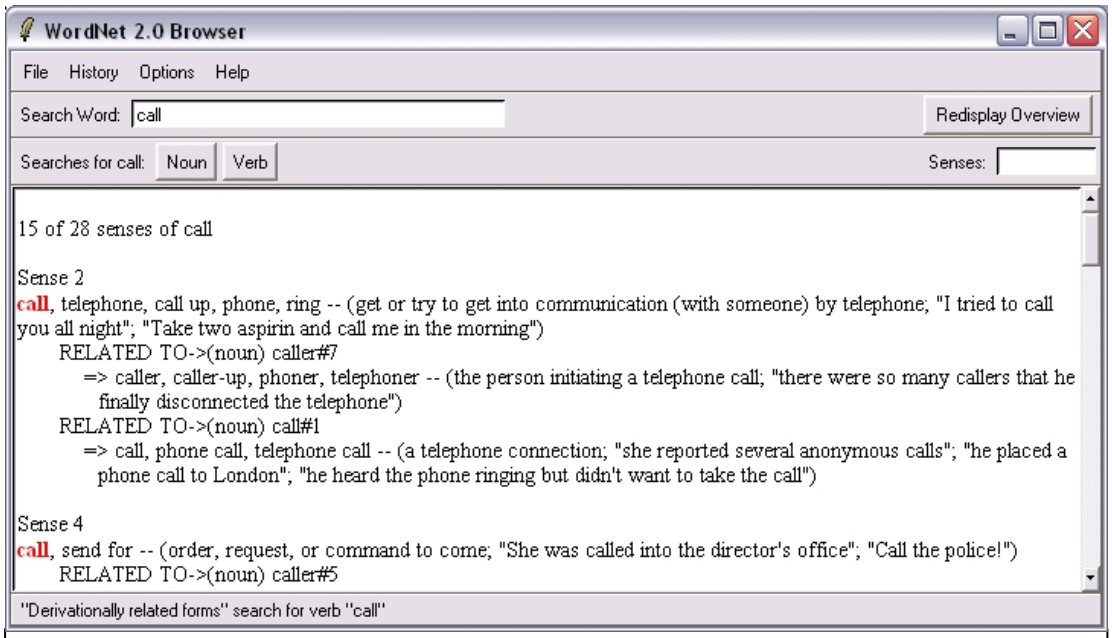

Figure 7.5. Some derivationally related forms of the verb “call”.

Sense 2 of “call” shows a relation to sense 7 of the noun “caller”. What this means is that in the sentence “x called y”, x is the “caller”. This was very interesting for us, as we saw that it would be possible to semantically analyze a sentence with this type of information. The project we decided to do we therefore called *Semantic Analyzer*. Sentences which contained an “actor” and “receiver” were to be analyzed (which we called actor-receiver scenarios). Take this sentence for example:

(1) The wife called the plumber

In this case, we see the wife as an “actor” as she is the one who does something, i.e. calls. The plumber is the “receiver”, since he/she is at the receiving end of the actors doing, i.e. receives a call.

The analyzing is done by looking at the hypernym trees for the actor and for its derivationally related form. In sentence (1) the actor is “wife” and the derivationally related form is “caller”. The hypernym trees are shown in Figure 7.6 and 7.7.

7.1.1 Problems encountered

A few problems arose during the development of the Email Extractor PR. The problems were:

- **GATE requiring an older version of Java SDK**: At first, GATE did not start on either of the machines we tried it on even though they advertised it to run on "Java version 1.4 or above."[20] We both needed to downgrade to 1.4.1 from version 1.4.2.

- **GATE locked the PR**: When we informed GATE to load the processing resource it did so while at the same time locking it, making it impossible to recompile the project without restarting GATE. Restarting GATE every time we wanted to fix a bug became frustrating.

- **Assigning a document to the PR**: We found it rather tiresome to assign a document to the processing resources. The only way to do it was by selecting the PR's one by one and choosing the working document. It would have been advantageous to be able to select all the PR's and choose their working document in one single step.

Although we did not have that many problems with gate during this experiment, having to repeat certain tasks over and over again certainly lowers the productivity.

7.2 A GATE project incorporating WordNet

After having done the basic project with e-mail extraction we wanted to do something a little bit more advanced. We decided to create a program which would incorporate WordNet, in order to cover one more fundamental concept. After having done some more research on how WordNet functioned, we found an interesting feature called "*Derivationally related forms*" which showed relations between words. For example, when looking up "call" as a verb and then viewing its derivationally related forms, a number of different senses for these forms are presented, as is shown in Figure 7.5.

[20] This can be found in the user documentation from http://www.gate.ac.uk.

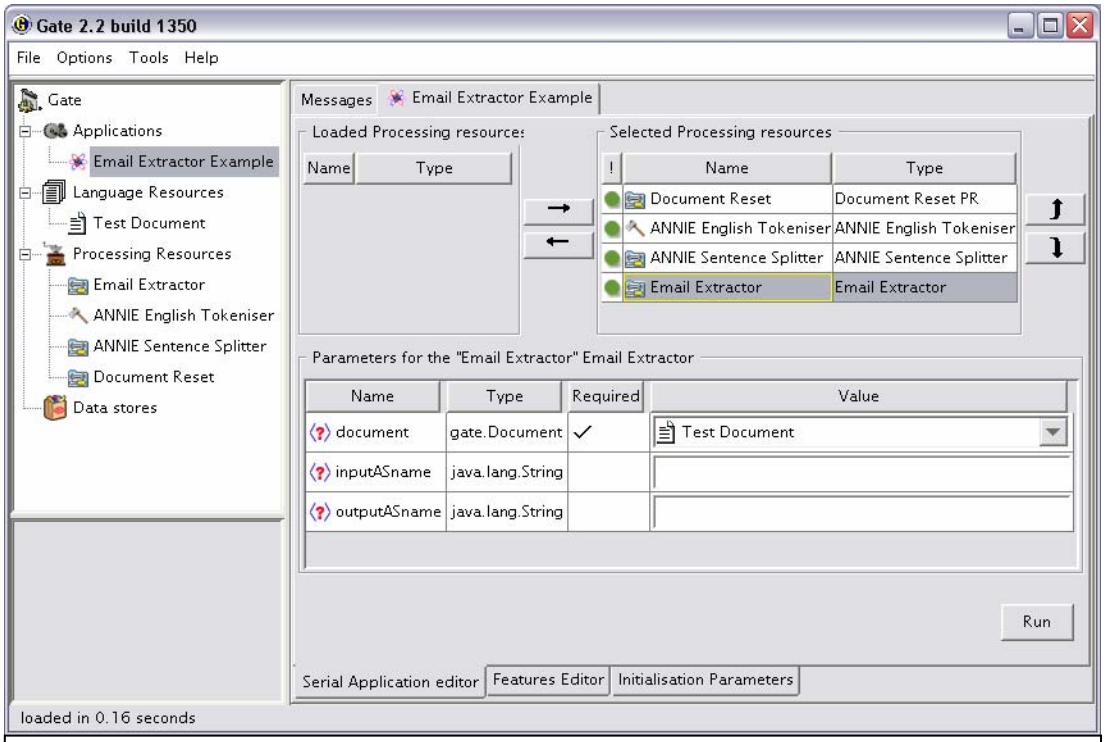

Figure 7.3. The Email Extractor is added to the pipeline, and we are ready to go.

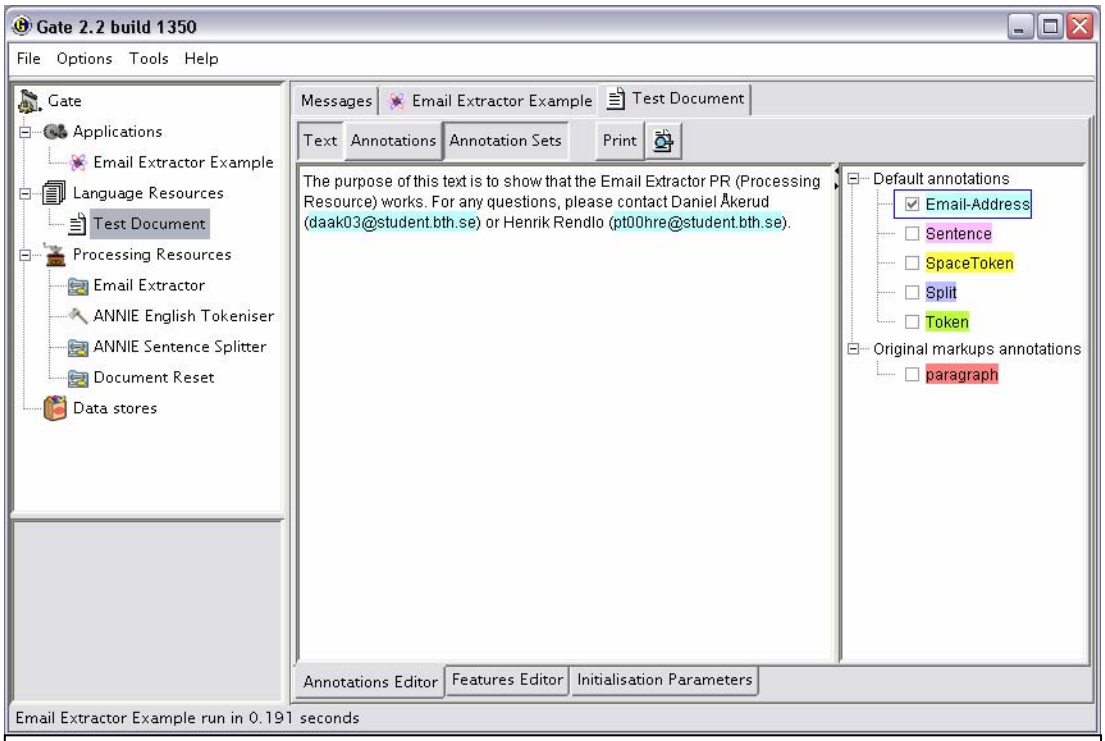

Figure 7.4. The Email Extractor has marked the email addresses in the document.

```
(                 // Start of email group.

[a-zA-Z0-9]+      // One or more of the allowed characters.

(\.[a-zA-Z0-9]+)* // Punctuation, and then one or more allowed
                  // characters. all this, zero or more times.
@                 // The obligatory @ character.

[a-zA-Z0-9]+      // One or more of the allowed characters.

(\.[a-zA-Z0-9]+)* // Punctuation, and then one or more allowed
                  // characters. all this, zero or more times.
\.[a-zA-Z]{2,4}   // The domain name.

)                 // End of email group.

([^a-zA-Z]|$)     // Used to find end of email address. Either not
                  // an email character, or end of search string.
```

Figure 7.2. Explanation of the regular expression used to get email addresses from documents.

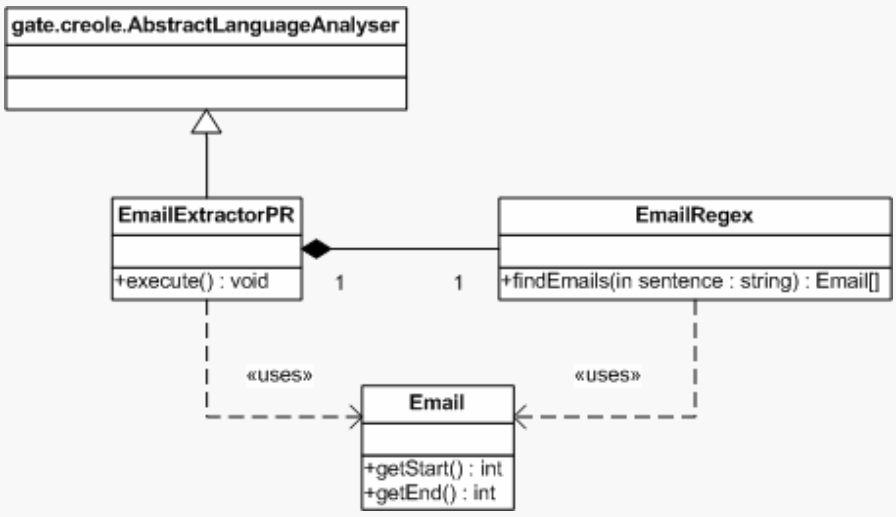

Figure 7.1. This is a UML diagram over the Email Extractor PR.

The three classes that the system consists of are:

- **EmailExtractorPR**: This class is the GATE processing resource. It handles the most important tasks, such as extracting sentences from the document's annotation-set, and adding new e-mail annotations when an e-mail is found.

- **EmailRegex**: This class contains a regular expression designed to find e-mails. The findEmails method takes a sentence as a parameter. The output is an array of Email instances used by the EmailExtractorPR to mark the e-mails in the document.

- **Email**: This class represents an e-mail found by the EmailRegex class. It has two methods, getStart and getEnd that returns the start and end position of the e-mail relative to the searched sentence. Recall that this information is needed in the referential approach, as the notations store the start and end positions of the text that is referred to.

The EmailRegex class contains a regular expression designed to find e-mails, and a simple algorithm to apply this to sentences. The regular expression is presented and explained in Figure 7.2, and examples of what it looks like in GATE is illustrated in Figure 7.3 and 7.4.

7 AN EMPIRICAL EVALUATION

In order to certify that the literature study covered relevant parts, and to give a more practical view of the development of NLP applications, we decided to make two different projects covering different aspects of NLP. After some brainstorming we came up with two projects:

- **A basic GATE project**. To get started we decided to create a basic project. This would allow us to locate the fundamental concepts we had written about in our literature study and become familiar with the most basic functionality within GATE.

- **A GATE project incorporating WordNet**. When the goals of the basic project had been achieved, we wanted to delve deeper and also incorporate WordNet. To recall, WordNet is a computational lexicon discussed in Chapter 3.2.2. The purpose of this project was to get a practical view of one more fundamental concept.

We will now describe how we tackled the first project.

7.1 A basic GATE project

Since we were not so familiar with GATE in the beginning, we wanted to try out a small, yet meaningful, example. We came up with an idea of a processing resource, named *Email Extractor*, capable of marking all emails in a document. This is a very basic example of information extraction – one thing that GATE was designed to work with. Just to emphasize, the main focus will not be on the e-mail extracting part, but rather to locate certain fundamental concepts and become familiar with basic functionality.

The next step was to design the system. Figure 7.1 is a UML diagram illustrating the Email Extractor Processing Resource.

6.3 ANNIE

GATE contains a set of built in CREOLE resources called *ANNIE* (A Nearly New Information Extraction system), ready to be used in a new application [Cunningham et al. 2002a, pp. 102-107]. They are very generic, and give a good start in the development process. The ones we took advantage of in our empirical evaluation are:

- **Document Reset**: This removes all the old annotations from the document.

- **English Tokeniser**: This processing resource splits text into tokens. Two new annotations are created: *Token* and *SpaceToken*. Both can have a feature called type. For instance, a token can be of type “word”, “punctuation” or “dashpunct” (“-“).

- **Sentence Splitter**: With the help of the tokeniser, this module splits the text into sentences. Two new annotations are created: *Sentence* and *Split*. “Sentence” contains the sentences, and “Split” contains the tokens that separate the sentences.

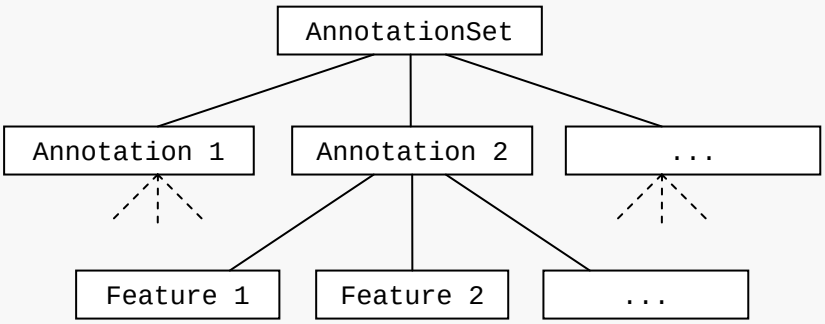

Figure 6.1. An illustration of the AnnotationSet → Annotation → Feature hierarchy present in GATE.

6.2 Components

GATE uses a component based architectural model where each component is called a *CREOLE* (a Collection of REusable Objects for Language Engineering) resource. For the new version of GATE, version 2.2, the *Java Beans*[19] component model is used. There are many different definitions of a component, but the most often used today is Szyperski's [Crnkovic 2003, p. 12]:

> *A software component is a unit of composition with contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*

There are three different types of components, namely [Cunningham et al. 2002b, p. 2]:

- **Language Resource (LR)**: Modules that contain only data, such as corpora and lexicons, are called *language resources* in GATE. These have previously been referred to these as declarative modules (see Chapter 3.2).

- **Processing Resource (PR)**: Modules that are algorithmic, for instance POS taggers and grammar parsers, are called *processing resources* in GATE. These have previously been referred to as procedural modules (see Chapter 3.3). In both our experiments, presented in Chapter 7, we have created GATE PR's to do the work.

- **Visual Resource (VR)**: These are resources that are built to visually represent some information or interact with the user in the GATE GUI.

[19] Java Beans are Java classes that obey certain interface conventions [Cunningham et al. 2002a, p. 54]

6 GATE

The team behind GATE refers to the system as being “an architecture, a framework and a development environment for LE (Language Engineering)” [Cunningham et al. 2002b, p. 1]. The latest version at the time of writing, version 2.2, is developed in Java (in contrast to older versions which used C++), and is available for a multitude of platforms including Windows, Linux and UNIX.

How GATE works will be the subject of this chapter, especially topics related to our experiments in order to keep it within boundaries of this thesis.

6.1 Text Information Management

As mentioned earlier, GATE uses the referential approach when it comes to text information management – an idea borrowed from the TIPSTER[16] project. Sponsored by DARPA, TIPSTER was a research effort focusing primarily on document detection, information extraction and summarization. TIPSTER was the pioneering project in using the referential approach for text information management [Grishman 1995]. Development of TIPSTER seized in 1998 for an unknown reason. The input to the GATE system can either be a single document or a set of documents, also known as corpus. Each document can have a set of annotations, an *AnnotationSet*. Each AnnotationSet can contain zero or more annotations. For instance, an AnnotationSet of type “Sentence” can contain any number of annotations describing where a sentence starts and ends. Each one of these annotations contains any number of features, each describing the annotation. For example, an annotation of type “Sentence” can have a feature containing how many times a certain word occurs in a sentence. Internally this is organized in directed acyclic graphs [Cunningham et al. 2002a, p. 69]. This hierarchy is illustrated in Figure 6.1. We have taken advantage of this system for both of our experiments. When it comes to storage of annotations, GATE has the possibility to store annotations either in a separate file, in an Oracle[17] database or in a PostgreSQL[18] database.

[16] TIPSTER, http://www.itl.nist.gov/iaui/894.02/related_projects/tipster/, last viewed 2004-05-04
[17] Oracle, a leading commercial database manager, http://www.oracle.com, last viewed 2004-05-04
[18] PostgreSQL, an open source database manager, http://www.postgresql.org, last viewed 2004-05-04

Courseware, aimed at teaching students NLP. The project is still in development, having its last release in March 2004.

5.1 Choosing a framework or toolkit

In order to give software engineers a better understanding of how to develop an NLP application and to give more concrete connections to the core concepts described in the literature study, we have chosen to conduct an empirical evaluation with a framework or toolkit. This part will explain our choice of such a framework or toolkit.

We immediately dismissed MULTEXT and ALEP since their development had seized. NLP is a very active field and therefore up-to-date developing environments are preferred.

After some though, we also dismissed LT-XML because it was focused solely on the manipulation of XML documents. Although essential to NLP development, more especially when using the additive approach for text information management, it is not of much help according to us because we seek a system that can offer a more complete solution.

We were left with two strong candidates – GATE and NLTK – which both possess desirable characteristics: (1) Some or all elements are GUI based, (2) both are active, ongoing projects, (3) they are well documented (as in thorough user documentation and published papers) and (4) both encourage reusability which is something strived for in software engineering.

However, NLTK possesses some undesirable characteristics. An important one was the early design goal of providing teachers with a pedagogic set of tools in teaching students NLP. Although doubtlessly useful, with this design decision followed some requirements that that made the system overall less pleasing in our eyes. First of all we were totally unfamiliar with Python, the language of choice. Despite the much discussed shallow learning curve, it is still a curve, and we were both very familiar with Java that was the language of choice in GATE. If we were not under time pressure, we would gladly have learnt a new language. Another result of this was that of efficiency. They state that “The toolkit does not need to be highly optimized for runtime performance. However, it should be efficient enough that students can use their NLP systems to perform real tasks.” [Loper 2004, p. 2].

Maybe the most important observation was that when we installed NLTK, we got the impression that it was not as suitable for creating complete NLP systems as GATE, but was rather a collection of small parts each explaining a specific NLP concept such as a POS tagger or a grammar parser. This is undeniably perfect in a pedagogic purpose which officially is the toolkit’s primary aim.

For us these are all acceptable decisions for a system used in an environment where the pedagogic aspects should permeate the system, but it is less pleasing for us.

Bringing this chapter to a close, our research led us to conclude that GATE is currently the most suitable choice for constructing a system with NLP technology.

5 FRAMEWORKS AND TOOLKITS

During the literature study, different NLP frameworks and toolkits were mentioned in various articles. Each of these trails were followed and investigated. Some trails led to a dead end, i.e. little or no information could be found about them. In addition to this, more than one significant attempt was made to find additional software and more information. It is of importance to mention that we sifted out projects with the goal of aiding in the development of NLP programs, rather than being a standalone program.

The different frameworks and toolkits will be presented now:

LT-XML[11]: LT-XML (made by the Language Technology Group, therefore LT) was formerly known as LT-NSL. Trying to find information about LT-NSL (Normalised SGML Library) did not result in much information. The reason for this was because it turned out that LT-NSL had changed name into LT-XML. LT-XML consists of a number of C applications for manipulating XML documents. It is also possible to write C applications for manipulating XML documents yourself using the provided API. Last major release was in September 2000, and last minor release was in April 2004.

GATE[12]: Funded by the EU and the U.K. Engineering and Physical Sciences Research Council (EPSRC), General Architecture for Text Engineering (GATE) is an infrastructure for developing various NLP applications. It consists of an architecture, a framework and a graphical development environment for NLP systems. The project is still in development. Features are constantly added to GATE (recently PostgreSQL and WordNet support, amongst others).

ALEP[13]: Initiated by the European Commission in 1994, ALEP (Advanced Language Engineering Platform) aimed at providing a general purpose environment for natural language experimentation. Development seized in 1995, and the reason can be traced back to the complexity of the product, as ALEP was intended for either skilled researchers or teams of researchers.

MULTEXT[14]: Abbreviation for Multilingual Text Tools and Corpora, MULTEXT is a project aimed at developing a set of standards and tools for analyzing and manipulating text corpora in multiple European languages. The project uses SGML for text information management. When it comes to concrete material, a few Unix-based command-line tools were found (and all of them were test versions) during research on a page that has been under construction since 1996. Further investigation revealed that that same year development seized for unknown reasons.

NLTK[15]: NLTK (Natural Language Toolkit) consists of several Python libraries and applications for developing NLP applications. The project is advertised as being

[11] LT-XML, http://www.ltg.ed.ac.uk/software/xml/, last viewed 2004-05-04
[12] GATE, http://gate.ac.uk/, last viewed 2004-05-04
[13] ALEP, http://www.iai.uni-sb.de/iaide/en/alep.htm, last viewed 2004-05-04
[14] MULTEXT, http://www.lpl.univ-aix.fr/projects/multext/, last viewed 2004-05-04
[15] NLTK, http://nltk.sourceforge.net/, last viewed 2004-05-04

and a Bayesian classifier based weight estimation algorithm. In [Choi 1999b], the author conducts experiments to evaluate the feasibility of the method, using a simple example that distinguishes verbs from nouns.

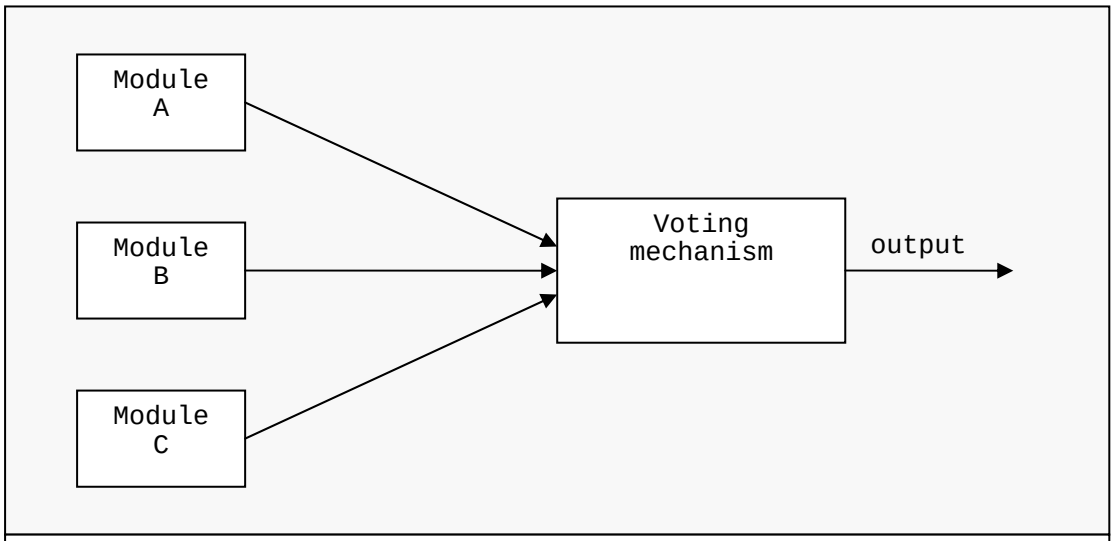

Figure 4.4. Simple voting applied to three modules.

In this example, each module is only given one vote. The voting mechanism could be improved by giving a weight to each module that corresponds to how well the module performs in isolation. This could be how well it performs overall, or how well it performs on the specific word [van Halteren et al. 1998]. For information about more advanced voting mechanisms in practical experiments, please consult [van Halteren et al. 1998] and [Brill and Wu 1998].

Eric Brill and Jun Wu conducted an experiment using four different POS taggers: Unigram, N-gram, Transformation-based and Maximum-entropy. An interesting early conclusion made was that the errors the different POS tagger made were not of the same type, increasing the chance that combinations of the taggers would achieve better results. If all the errors would have been of the same type, the tagger would not be able to complement each other. In the end, the best results of their experiment showed an increase in the success rate from 96.8% to 97.2% compared to their best tagger (Maximum-entropy).

In another, a little bit more extensive experiment conducted by van Halteren et al., four POS tagging modules where combined in different ways and their results where calculated using different voting mechanisms. The taggers where: N-gram, Transformation-based, Memory-based and Maximum-entropy. All sets of taggers gave better result than any subset of that same set. For example, combining A and B always yielded better results than either A or B separately. The best result where achieved using all four taggers, which increased the success rate from 97.43% to 97.92% compared to their best tagger (Maximum-entropy) [van Halteren et al. 1998]. This result is very much alike that in the experiment by Eric Brill and Jun Wu, both showing an increase in about a half percentage point.

In a paper written by Choi, TEA (Text Engineering Architecture) is described as being a “flexible architecture for developing and delivering platform independent text engineering (TE) systems” [Choi 1999a, p. 615]. Even though it is hard to find any sign of its use, according to the paper it allows for the construction of systems where modules are combined and output is chosen based on voting. Three different voting mechanisms are implemented, namely weighted average, weighted maximum

4.3 Whiteboard

A whiteboard architecture is a variation of a blackboard architecture. Instead of a shared data structure, there is a process called a *whiteboard*. Instead of knowledge sources there are *components*. Also, instead of one control module, each component has a *manager* which acts as an interface to a *coordinator* which interfaces with the whiteboard. This is illustrated in Figure 4.3. Another important difference between a blackboard and whiteboard architecture is that only the final and the most important intermediate data is written to the whiteboard. This means that the communication load is not as big as in a blackboard system [Lohuizen 1998, p. 7] [Boitet and Seligman 1994].

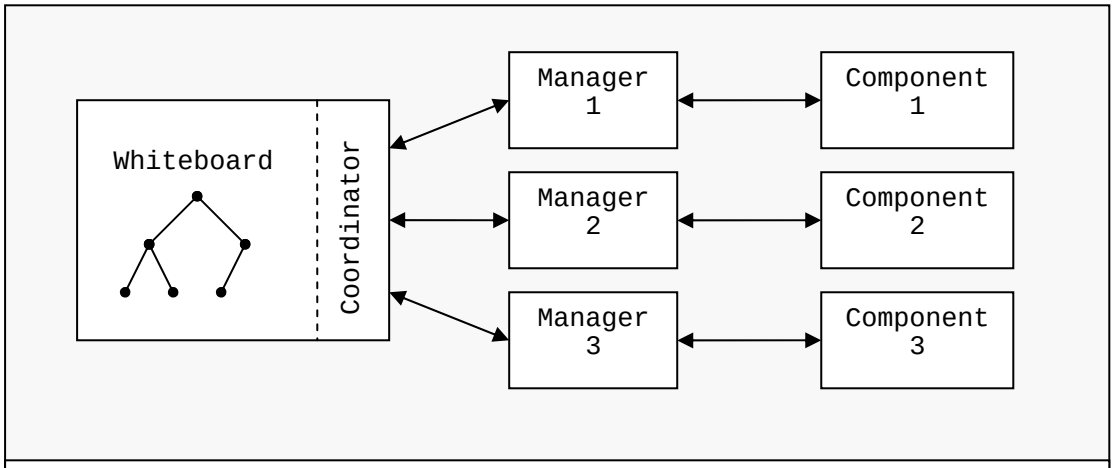

Figure 4.3. Illustration of a whiteboard model. Like in the blackboard architecture, the tree structure in the blackboard model should be viewed as data which is inferred to solve a problem.

The architecture of a whiteboard system makes it more suitable for parallel processing as components do not have to be concerned with simultaneous access to the whiteboard.

4.4 Voting

It has been demonstrated that by combining more than one module designed for a particular task, the overall effectiveness of the system can be improved [Choi 1999] [van Halteren et al. 1998] [Brill and Wu 1998].

For example, say that there exist three procedural modules – A, B and C – for POS tagging a word. Each module has the same goal, but tries to achieve it in different ways. If all three outputs from the modules are analyzed showing that module A and B conclude that a word is a verb, but the last module, C, think that the word is a noun, this would imply that the word is a verb. This type of voting mechanism is illustrated in Figure 4.4.

- **Lexical transfer MT**. This module simply uses a lexicon to translate text. Since this kind of translation is very basic, the module is only used as a last resort if none of the other modules could find a translation.

When each of the different MT modules has finished processing the input, it is the job of the *chart manager* to select the best translation of a text by merging the data from the MT modules.

Parallel architectures can either be basic, as discussed above, or they can be more complex, which will be discussed next.

4.2 Blackboard

The concept of a blackboard architecture was first introduced by Erman et al. in an NLP system, or more precisely, a speech-understanding system [Erman et al. 1980]. The idea of a blackboard architecture is to have modules, *knowledge sources*, which write all their intermediate information on a shared data structure, *a blackboard*, in order to find a solution to a problem. This is illustrated in Figure 4.2. Knowledge sources respond to changes on the blackboard. This can lead to them adding data to the blackboard or reading data so that new knowledge can be inferred and placed on the blackboard enabling other knowledge sources to use it [Nii 1986].

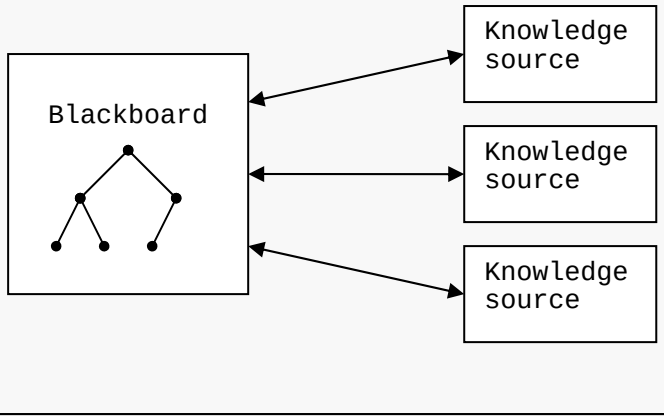

Figure 4.2. Illustration of a blackboard model. The tree structure in the blackboard model should be viewed as data which is inferred to solve a problem.

There also exist blackboard architectures where some sort of control module is present monitoring the blackboard for changes. Its job is to steer the knowledge sources into the right direction and control the order of computation, if needed [van Lohuizen 1998, p. 6] [Liere et al. 1998, p. 225] [Hewett and Hewett 1993].

When it comes to NLP systems, blackboards are usually not needed since data often flows in one direction. However, using blackboard architectures would lead to a more open system and therefore easier to extend. It would also allow for easier implementation of a two-way flow of information [Lohuizen 1998, p. 6].

4 PARALLEL NLP ARCHITECTURES

In this chapter, parallel architectures of different complexities will be described, ranging from the most basic to more advanced.

4.1 Overview

In parallel architectures, there is no chain of modules which process data. Instead, modules process data separately and then the result is combined [Lohuizen 1998, p. 5].

An example of a parallel architecture is the Pangloss Mark III Machine Translation System. The system translates texts from Spanish to English. Its architecture is shown in Figure 4.1 [Nirenburg 1995, p. 5].

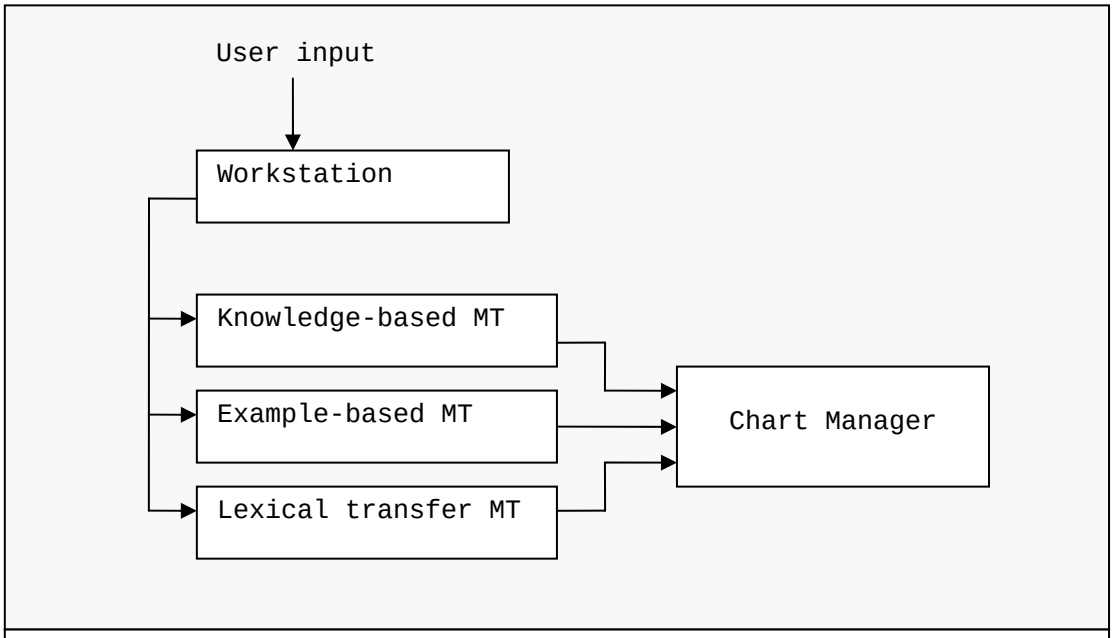

Figure 4.1. The Pangloss Mark III Translation system.

In the Pangloss system there are three different machine translation modules which are run independently and concurrently [Nirenburg 1995]:

- **Knowledge-based MT**. This module uses rules to create possible representations of clauses (parts of complex sentences), and ranks them according to their likelihood.

- **Example-based MT**. This module compares parts of an input text to a bilingual text archive (an archive containing a text in one language and a translation of that text in another language). The module then selects the closest match.

- **Format restrictions**: Systems using the additive approach benefits from something called Document Type Definition, or DTD for short. DTD restricts the looks of the documents and thus also restricts the allowed annotations. You can think of this as an interface, forcing developers to adhere to a standard. In contrast, as D. McKelvie state, “GATE allows any annotations anywhere” [McKelvie 1997, p. 232].

Looking at the above list, it is clear the number of points speaking in favor of the referential approach outweigh those speaking in favor of the additive approach. However, do not conclude that the referential approach is superior to the additive, since different scenarios require different solutions.

3.4.3 The abstraction approach

The abstraction approach differs greatly from both the additive and referential. The original text is abstracted using formalism such as *feature structure matrices* [Jurafsky and Martin 2000, p. 397], in which arbitrary information can be stored.

Abstraction approaches are rather complicate in nature, and the techniques involved are more related to natural language understanding (NLU) than NLP. Therefore, the abstraction approach is not discussed in further detail.

3.4.4 Comparison between the additive and the referential approach

The differences between the additive and referential approaches are many [McKelvie 1997] [Cunningham et al. 1997] [Cunningham 2000]:

- **Read-only media**: Using the additive approach, reading large texts from read-only media such as a CD, DVD or even an online resource demands a full copy of the original text. If it is big, let us say a large annotated corpus, this problem is not to neglect. The referential approach has no problem with this since it does not need to alter the original text.

- **Protection mechanisms**: In a system using the referential approach, things such as multi-level access (for example allowing write operations only for a certain user) is handled automatically by the database.

- **Concurrency control**: Again, using the referential approach, allowing multiple components to read and write annotations in parallel is controlled by built-in and well understood locking mechanisms in the database. In the additive approach, this would require additional implementation of complex concepts.

- **Parsing overhead**: For a system using the additive approach, input entering an arbitrary module needs to be parsed in order to extract relevant information such as the original text or information added by other modules. As documents grow, the parsing could become a bottleneck. LT-NSL has a workaround for this using something called normalized SGML [McKelvie 1997, p. 230]. Still, this solution does not scale as well as the referential approach.

- **Storage overhead**: Assume that two modules are to work on the same input in parallel. In an additive approach, each module need to work on one copy of the input each, where in a referential approach both modules can access the database independently of one another.

- **DBMS**: In the additive approach there is no need to have an external database (DBMS, or database management system) handling the references. Although this is not required in the referential approach, not using an external database eliminates two of the advantages discussed above, namely *Protection mechanisms* and *Concurrency control*.

```
<!DOCTYPE CESANA PUBLIC "-//CES//DTD cesAna//EN" []>
<CESANA VERSION="1.12">
<CHUNKLIST>
<CHUNK ID="C2" TYPE="DIV_R" FROM="1.2.1.1.2.2.3.2">
<PAR ID="C2P14" TYPE="P" FROM="1.2.1.1.2.2.3.2.14">
<S ID="C2P14S1">
<TOK><ORTH>—</ORTH><CTAG>-</CTAG></TOK>
<TOK><ORTH>the</ORTH><CTAG>AT</CTAG></TOK>
<TOK><ORTH>need</ORTH><CTAG>NN1</CTAG></TOK>
<TOK><ORTH>to</ORTH><CTAG>TO</CTAG></TOK>
<TOK><ORTH>ensure</ORTH><CTAG>VVI</CTAG></TOK>
<TOK><ORTH>the</ORTH><CTAG>AT</CTAG></TOK>
<TOK><ORTH>broadest</ORTH><CTAG>JJT</CTAG></TOK>
<TOK><ORTH>possible</ORTH><CTAG>JJ</CTAG></TOK>
<TOK><ORTH>geographic</ORTH><CTAG>JJ</CTAG></TOK>
<TOK><ORTH>representation</ORTH><CTAG>NN1</CTAG></TOK>
<TOK><ORTH>.</ORTH><CTAG>.</CTAG></TOK>
</S>
</PAR>
</CHUNK>
</CHUNKLIST>
</CESANA>
```

Figure 3.20. This is an example of an additive approach to text information management used in the MULTEXT system. The tag <orth> shows the orthographic (original) form of the word. The tag <ctag> (the “corpus tag”) is added by a POS tagger. There are of course more tags that can be

3.4.2 The referential approach

In contrast to the additive approach, architectures based on the referential approach separate the original text from the data about the text. The information is stored separately in, for example, a regular database manager or just a separate file.

Information is stored in what is called *annotations*. Annotations associate *attributes* with a specific piece of the original text, where attributes is arbitrary information [Cunningham et al. 1997]. Examples could be POS tags, grammatical structures, etc.

Figure 3.21 illustrate an example of storing annotations separated from the original text, and instead *referencing* it [Cunningham et al. 1997].

Id	Type	Span Start	Span End	Attributes
1	token	0	5	Pos=NP
2	token	6	13	Pos=VBD
3	token	14	17	Pos=DT
4	token	18	22	Pos=NN
5	token	22	23	
6	name	0	5	name_type=person
7	sentence	0	23	

Figure 3.21. Annotations generated for the sentence “Sarah savored the soup”.

5. Since a terminal now has been connected to the start symbol S, “the” is processed in the same way as “call” until a tree has been generated.

Step 1, 2 and 4 show the bottom-up approach of the parser, where the input is taken into consideration. Step 3 shows the top-down prediction, where “NP” is added to the tree.

In this chapter, only some basic algorithms have been discussed. Of course there exist more complex and efficient ones.

3.4 Text Information Management

Every module in an NLP system takes a certain input and delivers certain output. There are different ways to store this information between modules. That is, the information which is generated from the modules. There exist substantially different variants of how to handle this. This chapter aims at explaining, and emphasizing differences between two different approaches, namely the *additive* approach, the *referential* approach. For the sake of completeness the *abstraction* approach will also be mentioned [Cunningham et al. 1997] [Cunningham 2000].

3.4.1 The additive approach

Architectures based on the additive approach have one single stream of data containing both the original text and the output from the modules. After a module has done its processing, the input is bundled with the added data to create an output. This single stream of information grows between each processing step. This approach is also called *markup based*, referring to markup based languages such as SGML (Standard Generalized Markup Language) that has been the preferred choice.

Figure 3.20 shows an excerpt of a file taken from the MULTEXT system described in Chapter 5, which uses SGML for text information management. The example has the original text, token by token, in an <orth> (*orthographic*, or original form of the word) tag, and information from a POS tagger in a <ctag> (*corpus tag*) tag.

completely wrong terminals. In other words, it does not care for what the input is [Jurafsky and Marin 2000, p. 363].

Left-corner parsing

Since there are both advantages and disadvantages with the above approaches as mentioned in the comparison, a combination of them has been developed, which is called *left-corner parsing*. Left-corner means the left-most symbol on the right side of a rule, i.e. the left-most symbol in the rule "VP → Verb NP" would be "Verb". The idea of left-corner parsing is to alternate the bottom-up approach with the top-down approach [WWW NLPTP] [Nederhof 1993] [WWW LCA].

An example of left corner parsing is shown in Figure 3.19.

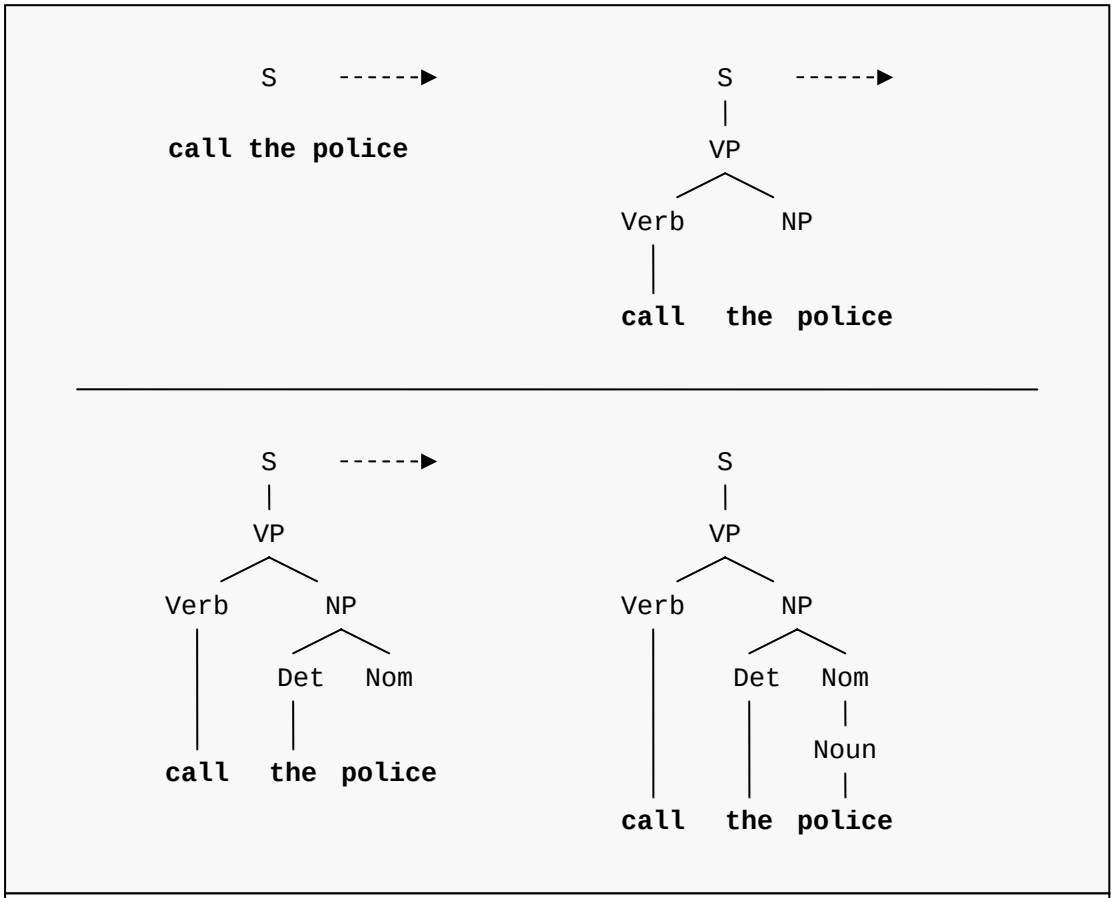

Figure 3.19. A left-corner parser expanding the tree to find out if "call the police" is a grammatically correct sentence. Some intermediate steps have been omitted.

An explanation of the first steps in the figure:

1. A rule where "call" is the left corner is selected. In this case "Verb → call".
2. A rule where "Verb" is the left corner is selected. In this case "VP → Verb NP".
3. Since the rule "VP → Verb NP" also has a "NP" it therefore has to be added to the graph.
4. A rule where "VP" is the left corner is selected. In this case "S → VP".

Bottom-up parsing

This is the opposite of top-down parsing, as it starts at the terminal symbols. The grammar is used for parsing until a complete tree has been created with a start symbol. An example of this is shown in Figure 3.18.

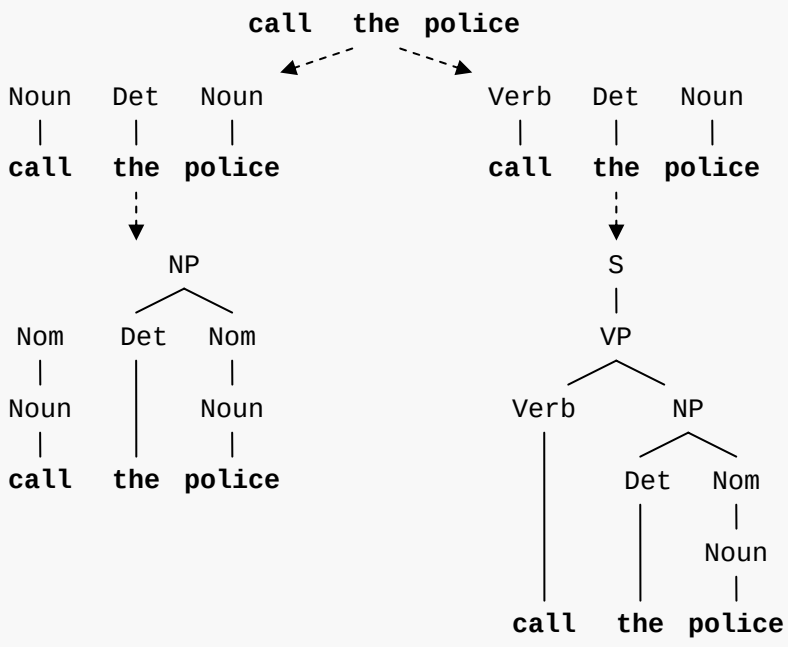

Figure 3.18. A bottom-up parser expanding trees to find out if "call the police" is a grammatically correct sentence. Some intermediate steps have been omitted.

An explanation of the first steps in the figure:

1. The rule "Noun → call" is applied to "call".
2. The rule "Det → the" is applied to "the".
3. The rule "Noun → police" is applied to "police".
4. The rule "Nom → Noun" is applied to "Noun".
5. This continues until no more rules can be applied. Unfortunately, the result is not the correct tree so the search continues.

The example shows that the left tree neither is complete nor contains a start symbol, and is therefore not grammatically correct. The right tree however fulfills both those requirements and therefore confirms that "call the police" is a grammatically correct sentence.

Comparison of top-down and bottom-up parsing

The main difference between the two strategies is that the top-down parser always creates a syntactically correct tree as it begins with the start symbol S, while the bottom-up parser might create a syntactically incorrect tree. Because of this, the top-down parser does not waste time exploring sub trees that are syntactically incorrect, which the bottom-up parser does. On the other hand, since bottom-up approach starts from the terminals, it does not spend time creating trees which might have

```
Mini grammar
S        →  VP
VP       →  Verb NP | Verb
NP       →  Det Nom
Nom      →  Noun | Noun Nom

Mini lexicon
Verb     →  call
Det      →  the
Noun     →  police | call
```

Figure 3.16. A mini grammar and lexicon.

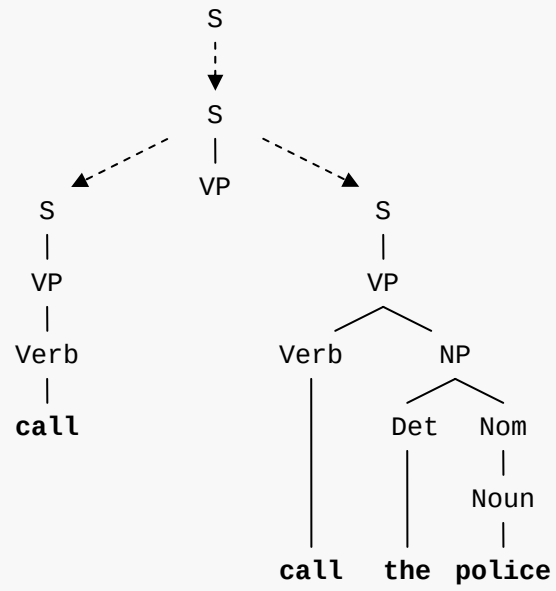

Figure 3.17. A top-down parser expanding trees to find out if "call the police" is a grammatically correct sentence. Some intermediate steps have been omitted. Note that since the grammar used is not a complete grammar of English, even a sentence such as "call" is correct.

An explanation of the first steps in Figure 3.17:

1. The rule "S → VP" is applied to "S".
2. The rule "VP → Verb" is applied to "VP".
3. The rule "Verb → call" is applied to "Verb".
4. One tree has been created which does not match the sentence "call the police" and therefore other rules are applied to "S" to continue the search for a tree which can fit the input sentence "call the police".

The example shows that "call the police" is a correct sentence using the grammar in Figure 3.16, since the rules specified allow for sentences in the form of "Verb Det Noun".

This show that it is much more likely that race is a verb than a noun. Now the two probabilities are multiplied:

$$P(NN \mid TO) * P(race \mid NN) \approx 0.000007$$

$$P(VB \mid TO) * P(race \mid VB) \approx 0.00001$$

The results are very interesting. Even though the probability of "race" being a noun was much greater than it being a verb (about 14 times more likely), the results show that it is most probable to be a verb. The reason for this is because the tag before "race" was taken into account, and the probability of "to" in front of a verb was greater (about 15 times more likely).

As mentioned before, the previous example calculated the most probable tag for *single* word (unigram tagger) A real HMM tagger calculates the best *sequence* of tags for a sentence (n-gram tagger).

Transformation-based tagging

Transformation-based tagging uses the ideas behind both rule-based and stochastic tagging. The first step in transformation-based tagging is to mark each word with its most likely tag. If the word is not known, the most likely tag is guessed based on certain properties of the word. After this, rules which take the context into account are applied in order to change mistagged words. [Brill and Wu 1998, p. 192]. An example of mistagging could be that the word "race" first is tagged as a noun because the context is neglected. Then a rule is applied concluding that race is actually a verb.

3.3.2 Grammar parser

This chapter will discuss three different parsing algorithms which are used for parsing trees. First the two most basic will be explained, followed by a comparison between them highlighting the pros and cons. Then a third will be explained which combines the two basic algorithms.

Grammar parsers play an important role in many applications [Jurafsky and Martin 2000, p. 358]. A common practical use of a grammar parser is for grammar checking in word processors, which can be found in word processors. The idea is to find a tree which fit the input by using a grammar and a lexicon. There exist many algorithms for parsing trees. The two most basic algorithms are the *top-down* and the *bottom-up* algorithms which will be explained now [Jurafsky and Martin 2000, p. 360] [WWW NLPTP]:

Top-down parsing

A top-down parser starts from the start symbol S, and works its way down to find a tree (or trees in case of an ambiguous sentence) that can fit a specified sentence. If it succeeds, the sentence is grammatically correct. Figure 3.17 shows an example of how a top-down parser could work, by expanding the trees using the rules in Figure 3.16.

Martin 2000, p. 303]. The sentences have been taken from the Brown Corpus and shortened:

*(1) Secretariat/NNP is/VBZ expected/VBN to/TO **race**/VB tomorrow/NN*

*(2) People/NNS continue/VBP to/TO inquire/VB the/DT reason/NN for/IN the/DT **race**/NN for/IN outer/JJ space/NN*

In the above sentences "race" can either be a verb or a noun. For the sake of simplicity, it will be assumed that every tag is known except for the tag associated to "race". According to the above formula, we should be looking at the previous tag and the current word, so the rest of the tags are irrelevant in this case

(3) to/TO race/?

(4) the/DT race/?

The focus will now be on (3). The task is to calculate which tag is most likely for "race" to be; a noun (NN) or a verb (VB). The most likely tag is the formula that creates the highest value:

$$P(NN \mid TO) * P(race \mid NN)$$

$$P (VB \mid TO) * P(race \mid VB)$$

Explanation of the first probabilities: P(NN | TO) and P(VB | TO), are the probabilities of a noun (or a verb) being after the tag "TO". This can be calculated from a corpus by counting all how many instances there are of "TO" in front of a specific tag. The probability of "TO" being in front of a noun or a verb can then easily be calculated. Jurafsky and Martin calculated this combining two corpora and came up with the following results[10] [Jurafsky and Martin 2000, p. 304]:

$$P(NN \mid TO) \approx 0.021$$

$$P(VB \mid TO) \approx 0.34$$

The results show that it is much more likely that "TO" is in front of a verb than it is of being in front of a noun. This makes sense, as one sees "to run", "to walk", "to read" more often than "to work" (as in "go to work, lazy-bones!").

Explanation of the second probabilities: P(race | NN) and P(race | VB), are the probability of a noun or verb being "race". Jurafsky and Martin calculated these probabilities as well from the same two corpora [Jurafsky and Martin 2000, p. 305]:

$$P(race \mid NN) \approx 0.00041$$

$$P(race \mid VB) \approx 0.00003$$

[10] 0.021 and 0.34 sums up to 36.1%, leaving 63.9% for other word classes following the word class TO. For instance, the word class TO can also be followed by the word classes "determiner" (as in "I went to *the* race") or "pronoun" (as in "They sent a package to *me*").

```
Adverbial-That rule
Given input: “that”
if
      (+1 A/ADV/QUANT);       /* if next word is adjective, adverb
                              or quantifiers */
      (+2 SENT-LIM);          /*  and  the  following  word  is  a
                              sentence  boundary */
      (NOT -1 SVOC/A);        /* and the previous like “consider”
                              which allows adjectives as
                              object complements */
then eliminate non-ADV tags
else eliminate ADV tag
```

Figure 3.15. A sample from a rule-based part-of-speech tagger which function is to disambiguate the word “that”.

As mentioned, this is a simplified version of the ENGTWOL tagger. There are more rules which take the probability of a tag into consideration.

Stochastic part-of-speech tagging

A stochastic tagger makes its choice based on the probability of a word being a certain word class by using corpora. The most basic tagger of this kind is called *unigram*. A unigram tagger simply does not look at the context in which a word appears. It only looks at *one* word, hence its name. Despite their simplicity, such taggers have scored accuracies in the range of 90-94% [Brill and Wu 1998, p. 192]. There exist more complex algorithms which take the context into mind. These are called *n-gram* taggers [Brill and Wu 1998, p. 191]. There are different implementations of n-gram taggers, and we have chosen to describe that uses the Hidden Markov Model (HMM) due to the availability of comprehensible literature. An HMM tagger will choose the sequence of tags which maximizes the formula [Jurafsky and Martin 2000, p. 303]:

P(word | tag) * P(tag | previous n tags)

This formula is designed for choosing the best tag *sequence* for a sentence, not just a single tag for a word. This is what a formula for calculating the probability for a single tag looks like:

$$t_i = \underset{j}{\operatorname{argmax}}\, P(t_j \mid t_{i-1}) * P(w_i \mid t_j)$$

Symbol	**Meaning**
t_i	the most likely tag
argmax	the highest value is to be strived for
t_j	a tag to test
t_{i-1}	the previous tag
w_i	the current word

An example created by Jurafsky and Martin will be used to show how the above formula for calculating the most likely tag for a single word works [Jurafsky and

```
Can you do that for me?       // Auxiliary verb
A can of food                 // Noun
I often can my peaches        // Transitive verb
```

Figure 3.14. This is an example of a common ambiguous word in English.

The fact that many common words are ambiguous has been clearly shown in a study of the Brown Corpus. Although only 11.5 % of the word types (as in tags from the tagset) used in the Brown Corpus are ambiguous, more than 40 % of the total number of running number of words (also known as *tokens*) are ambiguous [Jurafsky and Martin 2000, p. 299].

At a first glance, disambiguating more than 40 % of tokens might seem as an overwhelming task. This is not the case however, since some tags are more likely than others to be correct. For example, even though "a" can either be a determiner (as in "A good book") or a letter (as an initial in "Robert A Field"), it is generally more likely that it is a determiner [Jurafsky and Martin 2000, p. 299].

The two most common tagging algorithms are *rule-based taggers* and *stochastic taggers* [Jurafsky and Martin 2000, p. 300]. These tagging algorithms will now be explained in more detail.

Rule-based part-of-speech tagging

The first version of a rule-based POS tagger first appeared in 1962. It was based on a two-stage architecture:

1. Each word was assigned every possible tag using a dictionary.
2. Hand-written rules were used for disambiguating the words so that each word only ended up with one tag associated to it.

The above steps are the same as the tagging algorithms of today use. However, today both steps are much more advanced. The lexicon produces exact lexical classes, and there are about 1,100 rules which can be applied to the input. Disambiguating works by looking at both which words are in front of and after a specific word, and the lexical classes for all those words. Figure 3.15 shows a simplified example of such a rule using the ENGTWOL[9] tagger [Jurafsky and Martin 2000, p. 302]. The goal of the rule is to disambiguate the word "that".

[9] The ENGTWOL tagger is a rule-based tagger which uses a two-stage architecture.

Chapter 3.2.1. This parse tree can be used to check for grammar mistakes [Jurafsky and Martin 2000, p. 358].

3.3 Fundamental Procedural Modules

This chapter will explain two fundamental procedural modules, namely the part-of-speech (POS) tagger and the grammar parser. Both are frequently used in NLP applications.

3.3.1 Part-of-speech tagger

Jurafsky and Martin define *part-of-speech tagging* as "…the process of assigning a part-of-speech or other lexical class marker to each word in a corpus" [Jurafsky and Martin 2000, p. 298].

A tagging algorithm uses a string of words and a tagset as input. The output is the string of words with the most likely tag assigned to each word. Figure 3.13 shows a sample from the ATIS[6] corpus using the Penn Treebank tagset [Jurafsky and Martin 2000, p. 299]:

```
VB    DT    NN       .
Book  that  flight   .

VBZ   DT    NN       VB     NN       ?
Does  that  flight   serve  dinner   ?
```

Figure 3.13. A sample from the ATIS corpus using the Penn Treebank tagset

As shown earlier, words can be spelled in the same way but still have different meanings (as with "bear" in chapter *Word classes*). "Book" is also such an ambiguous word. It can be a verb, as in "Book that flight". It can also be a noun, as in "I like to read a good book now and then". "That" has the same problems associated with it. It can be a determiner[7], as in "Book that flight". However, it can also be a complementizer[8], as in "I think that would be difficult". When these ambiguities occur, the POS tagger is responsible for the disambiguation of the words [Jurafsky and Martin 2000, p. 299].

With the above problem, it is good to know that most words in English are unambiguous, i.e. they can only have one tag associated to them. However, many of the most common words of English are ambiguous, as the sentences in Figure 3.14 show, in which "can" has different meanings.

[6] The Air Traffic Information System (ATIS) was a speech understanding system whose goal was to simulate helping a user book a flight by answering questions about potential airlines, times, dates, and so forth [Jurafsky and Martin 2000, p. 282].
[7] A determiner is a word in front of a noun which indicates the kind of reference the noun has, for example "a", "an", "the", "this", "some", and so on [WWW GoE].
[8] A complementizer is a word which links a verb together with its argument, as in "I thought *that* you might like some milk" [Jurafsky and Martin 2000, p. 293]. Another example is "I don't know *whether* he is happy or not".

A CFG consists of four parameters [Jurafsky and Martin 2000, p. 331] [Schmeiser and Barnard 1995]:

- **Non-terminal symbols:** These are symbols which are on the left side of a rule, for example `NP` and `Nom`.

- **Terminal symbols**: These are symbols which only appear on the right side of a rule, for example `Det` (determiner) and `Noun`.

- **Rules**: These are also known as productions. Rules describe how a symbol can be transformed to one or more symbols, for example `VP → Verb NP`.

- **A start symbol**: This is the start symbol `S`.

Parse trees, or in other words syntactic structures, can be represented graphically which is convenient for humans to read as shown in Figure 3.11. They can also be represented with brackets which is convenient for computers to read as shown in Figure 3.12 [Jurafsky and Martin 2000].

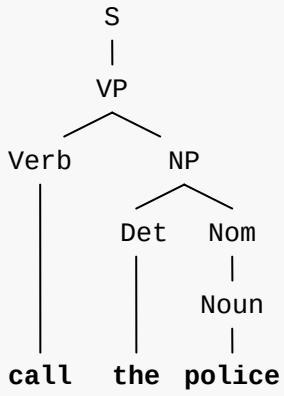

Figure 3.11. A graphical representation of a parse tree.

```
[S [Verb call] [NP [Det the] [Nom [Noun police]]]
```

Figure 3.12. A bracketed representation of a parse tree.

CFG:s are used for two main purposes [Jurafsky and Martin 2000, p. 327]:

- **Sentence generation**: A CFG can be used for generating new sentences. This can be of use when creating an abstract of a text sample, where the content of the text has to be re-written in a shortened form, as explained in Chapter 2.3.2.

- **Syntactical structure assignment**: Another use is for assigning structures to sentences. This is also known as syntactical parsing. When using CFG for assigning structure to sentences, a *parse tree* is created as explained in

3.2.3 Grammar

A grammar is a set of rules that defines the allowed syntax. In the computer world, a programming language follows a certain grammar defined by the compiler. Lex and Yacc are two programs that aid in the development of compilers and other *lexical* and *syntactical analyzers* [Niemann 2001, p. 3-4]. Lex and Yacc work together, where Lex (lexical analyzer) converts the input source to tokens, which serves as simplified input to Yacc (syntactical analyzer) [Niemann 2001, p. 6].

Yacc describe its grammar using a variant of the *Backus-Naur Form* (BNF)[4], as shown in Figure 3.7 and 3.8.

```
E  →  E + E
E  →  E * E
E  →  id
```

Figure 3.7. Example of a grammar

```
E  →  E * E
   →  E * z
   →  E + E * z
   →  E + y * z
   →  x + y * z
```

Figure 3.8. Example of an expression

Figure 3.7 shows a grammar consisting of simple rules for multiplying and adding numbers. Figure 3.8 on the other hand shows an expression generated from those rules.

The grammar of Yacc falls under level 2 of the Chomsky hierarchy[5] which is known as context-free grammars (CFG:s) [Salling 1998]. These types of grammars can also be used when creating a grammar for a natural language. Jurafsky and Martin define such a grammar as "a set of rules or productions, each of which expresses the ways that symbols of the language can be grouped and ordered together, and a lexicon of words and symbols" [Jurafsky and Martin 2000, p. 327].

Figure 3.9 and 3.10 show a set of rules and an expression generated by those rules. It is obvious that the grammar used in Yacc is very similar the examples below.

```
Mini grammar
S        →  VP
VP       →  Verb NP | Verb
NP       →  Det Nominal
Nom      →  Noun | Noun Nom

Mini lexicon
Verb     →  call
Det      →  the
Noun     →  police | call
```

Figure 3.9. Example of a mini grammar and lexicon.

```
S  →  VP
   →  Verb NP
   →  Verb Det Nom
   →  Verb Det Noun
   →  call the police
```

Figure 3.10. Example of an expression.

[4] A formal grammar for expressing context-free grammars.
[5] In 1959, Noam Chomsky presented four types of formal languages (regular grammars, context-free grammars, context-sensitive grammars and recursively enumerable languages), creating the Chomsky hierarchy [Salling 1998].

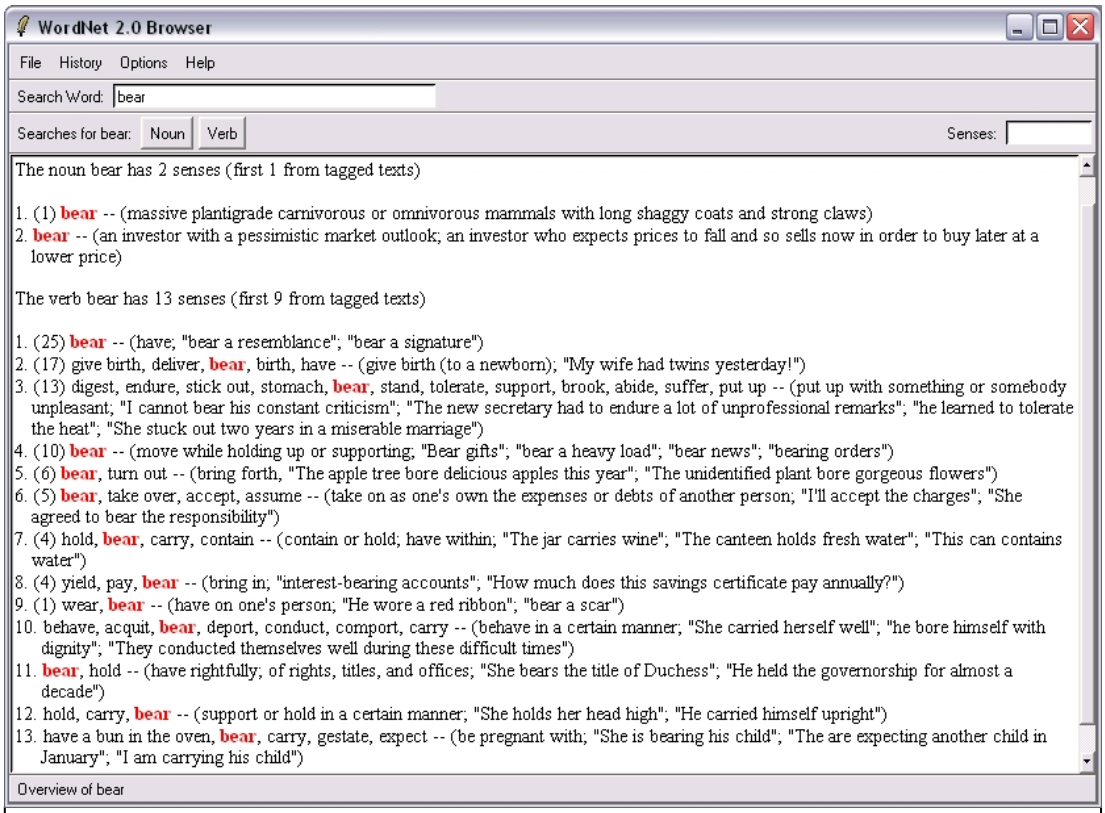

Figure 3.5. WordNet behaving much like a traditional lexicon.

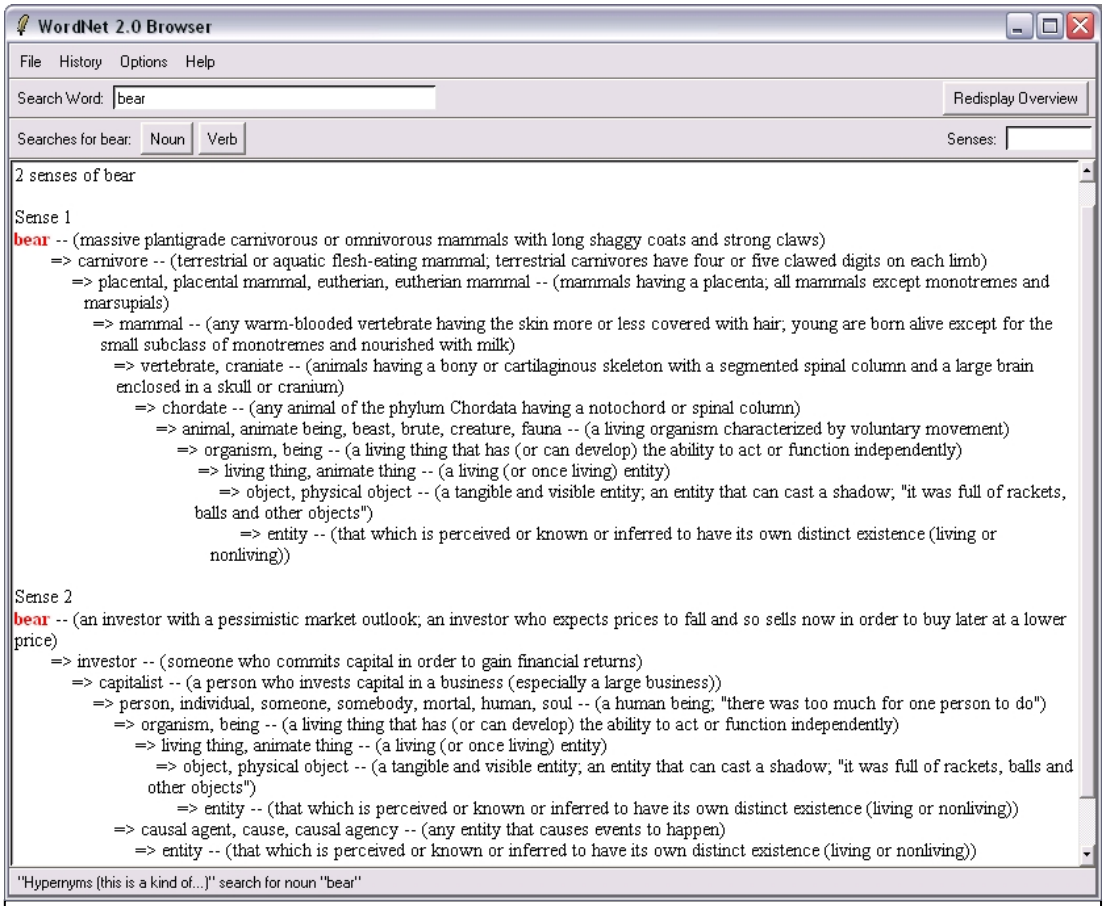

Figure 3.6. WordNet showing some of its true powers, displaying “Bear is a kind of…”

means investor because animal-bear and deer meet earlier in the semantic tree (in mammal), than person-bear and deer (in organism).

in the lexicon is called *lexeme*, and a lexicon can be seen as a finite set of lexemes [Jurafsky and Martin 2000, p. 590]. Examples of the usage will be described in Chapter 3.3, Fundamental Procedural Modules.

A problem with traditional lexicons is that they are not built for use in NLP. The problem however, is that traditional lexicons are not built for use in this area. M. Bates and R. Weischedel identifies that one of the current problems that need additional research in NLP is that of *knowledge acquisition*, which is the problem of gathering world knowledge and make it available to the computer. They state that the knowledge bases are too small, and acquiring new knowledge is too difficult [Bates and Weischedel 1993, p. 3]. There is however at least one known project that has taken the opportunity to help in solving this problem; WordNet.

WordNet is a well-known lexicon in the field of NLP. WordNet encompasses certain properties for words which make it ideal for use in this field compared to a traditional lexicon. The author of WordNet states [Miller 1995, p. 39]:

> *"[WordNet] links English nouns, verbs, adjectives, and adverbs to sets of synonyms that are in turn linked through semantic relations that determine word definitions"*

Lexicons available in machine-readable form are nothing new. However, WordNet is designed from the ground up to be used in a program. Another word for that kind of lexicon is a *computational lexicon*.

Figure 3.5 shows WordNet used like a traditional lexicon, while Figure 3.6 shows a more exotic property of WordNet - how the word "bear" is grouped with other words through semantic relations. Using WordNet, it is possible to retrieve information including (but not limited to):

- **Hyponyms**: WordNet can retrieve information about what other things that is of the same kind. For example, a search for "bear" would retrieve entries such as "grizzly bear", "brown bear", etc.

- **Holonyms**: This property retrieves information about words in a hierarchical manner. Searching for "homo sapiens sapiens", we can see that we are a primate somewhere down the tree, a mammal further down, and in the end a part of the animal kingdom.

- **Meronyms**: It is also possible to find out what a certain entity consists of. For example, searching for "computer" would give entries such as CPU, keyboard, memory, etc.

This is just a selected part of the views available for nouns, leaving verbs, adjectives and adverbs to be explored by the reader.

Suárez et al. describes how WordNet can be used in sense disambiguation [Suárez et al. 1999]. For example, if a computer were to analyze a text about "bear and deer", using the semantic structure shown in Figure 3.6, it is possible to rule out that "bear"

Differences between the Penn Treebank and the Brown corpora

The Penn Treebank *tagset*, that is the set of possible tags each word can get during the tagging process, is based on the Brown Corpus tagset. However, for various reasons the Penn Treebank team has tried to reduce the size of the tagset. They state that "A key strategy in reducing the tagset was to eliminate redundancy by taking into account both lexical and syntactic information", in other words *lexically or syntactically recoverable distinctions* [Marcus et al. 1993].

For example, pre-qualifiers (such as *quite*, *rather* and *such*) are tagged with the same tag as pre-quantifiers (such as *all*, *half* and *many*), but in the Brown corpus these are treated separately. This is a lexically recoverable distinction, since a rich lexicon such as WordNet (explained in Chapter #) can be used to recover for example that the word "many" is a quantifier.

Another problem with the large amount of tags in the Brown corpus is that the same words seem to be tagged inconsistently [Marcus et al. 1993, p. 4]. That is, even in the same context, the same word is not always given the same tag. This problem does not arise in the Penn Treebank corpus, because of the lower amount of details. To get the same amount of detail about the words in the Penn Treebank corpus, the user has to recover the information themselves.

Also, it is not always possible to assign a tag to a word with full confidence that it is right. In this case, Penn Treebank allows for more than one tag to be assigned to a specific word, separated by a "|" [Marcus et al. 1993, p. 5]. This is not implemented in the Brown corpus. An example is the text taken from the Penn Treebank corpus shown in Figure 3.4. In that text, "targeting" is either a form of a verb (VBG) or a noun (NN).

```
[ Martin/NNP Marietta/NNP Corp./NNP ]
was/VBD given/VBN
[ a/DT $/$ 29.9/CD million/CD Air/NNP Force/NNP contract/NN ]
for/IN
[ low-altitude/NN navigation/NN ]
and/CC
[ targeting/VBG|NN equipment/NN ]
./.
```

Figure 3.4. An example of a double tagged word in the Penn Treebank corpus.

The most significant difference is arguably the syntactical tree structure that is present in the Penn Treebank but not in the Brown corpus as shown in previous figures.

3.2.2 Lexicon

A *lexicon* is another fundamental concept in NLP. A lexicon is a collection of words bundled with information. The information, and amount of information, differs from lexicon to lexicon. Information commonly present includes part of speech labels (verb, noun, adjective), information describing the word and plural forms. One entry

- **Amount of data**. Choosing a large corpus increases the chance of having a good sample space, but it might take some time to process. In contrast, a smaller corpus is faster to process, but does not cover as much as a larger one.

- **Available corpora**. There are several corpora which are available for download. They can be free, need a user to register or cost money.

Two well-known corpora are the Brown Corpus and the Penn Treebank Corpus. The Brown Corpus is a 1 million word collection of samples from 500 written texts, in American English, from different genres (newspaper, novels, non-fiction, academic, etc.) which was assembled at Brown University in 1963-1964 [Jurafsky and Martin 2000, p. 166]. The Penn Treebank Corpus consists of about 7 million words with material such as IBM computer manuals and Wall Street Journal Articles also in American English [Taylor et al. 2001, p. 5]. Figure 3.2 and 3.3 show one sample from each corpus. Note that the text in the Penn Treebank corpus is not only tagged, but parts of it are also syntactically parsed in order to retrieve information about the syntactical structure of the sentences. A significant yet smaller part has both the tags and the syntactical structure in what they call a *combined* version.

```
``/`` I/ppss paint/vb the/at nothing/pn ''/'' ,/, he/pps said/vbd
once/rb to/in Franz/np Kline/np and/cc myself/ppl ,/, ``/`` the/at
nothing/pn that/dt is/bez behind/in the/at something/pn ,/, the/at
inexpressible/jj ,/, unpaintable/jj '/' tick/nn '/' in/in the/at
unconscious/nn ,/, the/at '/' spirit/nn '/' of/in the/at moment/nn
resting/vbg forever/rb ,/, suspended/vbn like/cs a/at huge/jj
balloon/nn ,/, in/in non-time/nn ''/'' ./.
```

Figure 3.2. A small sample from the Brown Corpus.

Raw

```
Pick a country, any country.
```

Tagged

```
Pick/VB
[ a/DT country/NN ]
,/,
[ any/DT country/NN ]
./.
```

Syntactically parsed

```
( (S (NP-SBJ *)
     (VP Pick
         (NP (NP a country)
             ,
             (NP any country)))
     .))
```

Combined

```
( (S
    (NP-SBJ (-NONE- *) )
    (VP (VB Pick)
      (NP
        (NP (DT a) (NN country) )
        (, ,)
        (NP (DT any) (NN country)
)))
    (. .) ))
```

Figure 3.3. Samples from the Penn Treebank Corpus.

- **Valid representation**: It is not feasible to have a collection of all possible utterances in a language, and therefore a sample has to be taken. In order for the sample to be of good use, it must be a valid representation of the language.

- **Finite size**: Although there exist corpora which continually grow in size, it is most common that they do not. The main advantage of a finite sized corpora is that they are a reliable source of quantitative data (the size of the corpora does not vary, and various measures remain unchanged) and that they have been more thoroughly sampled.

- **Machine-readable form**: These days most corpora are in machine-readable form, instead of in written or spoken form, which makes it possible to search and analyze them quickly using a program.

- **Standard reference:** Many corpora are used as a standard reference by researchers. One advantage is that differences in studies using the same corpus would mean that they really are different due to the methodology and not due to the data.

After having gone through what a corpus is and what it is used for, the process of creating a corpus will now be addressed.

Corpus Compilation

Corpus compilation is the process of combining texts to create a larger body of text. Which texts to choose depends on what the corpus is supposed to represent. If the corpus is to be focused on medicine, then it would be preferable to use texts from the medicine domain. If the corpus is supposed to represent the general language, it would be preferable to use texts from more than one domain. The process can be made in three different ways [WWW CL1]:

- **Randomized**: A randomized process is exactly what it sounds like. Texts are selected randomly without any thought on their usefulness.

- **Randomized with control**: An example could be that texts are divided into categories and then randomly X % is taken from the sports category, Y % from the nature category, and so on.

- **Controlled**: Texts are read through and the most useful texts are selected and added to a corpus without any involvement of randomization.

Choosing Corpora

There are several things to keep in mind when choosing corpora [WWW CL1]:

- **Kind of material**. This is medium (written, spoken or both), text type (newspaper, fiction, spoken discourse, scientific study, etc) and time (produced in the 18th century, 19th century, 1990's, etc).

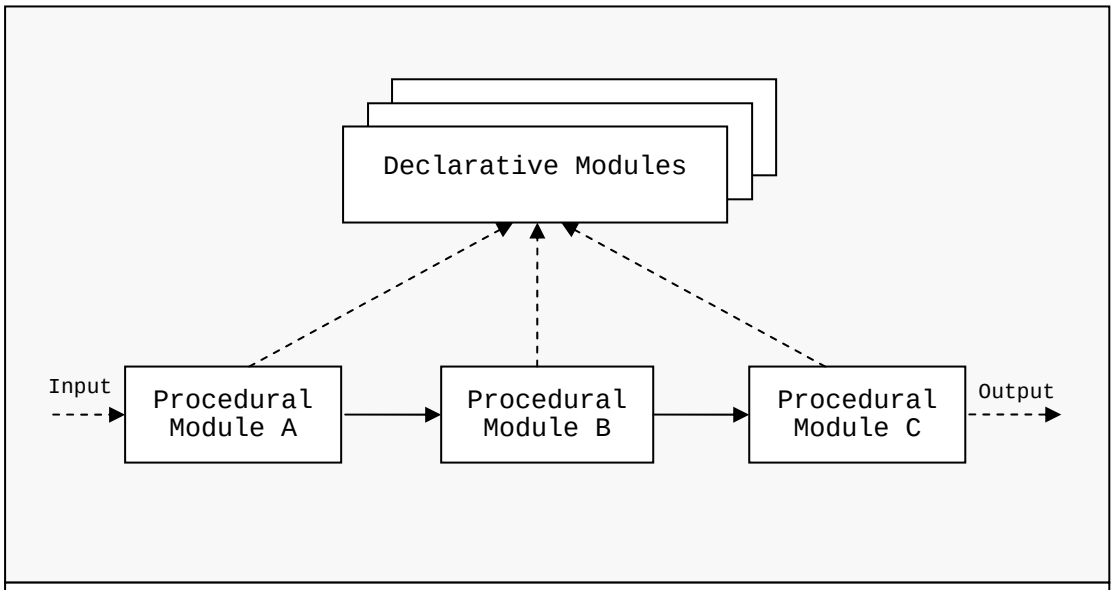

Figure 3.1. Illustration of the relations between declarative and procedural modules. The procedural modules use the information residing in the declarative modules to carry out their tasks.

The procedural modules use the declarative modules to generate output which is sent to a subsequent procedural module. Fundamental declarative modules are the subject of Chapter 3.2 (coming up next), and fundamental procedural modules are the subject of Chapter 3.3.

3.2 Fundamental Declarative Modules

In this chapter three very important core concepts of NLP will be discussed, namely corpora, lexicons and grammars. The concepts will be explained from a practical perspective, by using real-world samples of existing corpora and lexicon and presenting excerpts from different types of grammars.

3.2.1 Corpora

In the field of NLP, *corpora* (corpus in singular) are large collections of text, either in written or spoken form [Jurafsky and Martin 2000]. Corpora are used for solving ambiguities in texts and speech using statistical measures [Jurafsky and Martin 2000, p. 194].

Each word in such corpora can be marked with word classes and punctuation (comma, semicolon, etc), which is a process called *tagging* [Jurafsky and Martin 2000, p. 288]. Corpora can also be *syntactically parsed*, which creates a tree structure showing the internal organization of sentences. These two processes are used for analyzing naturally occurring texts. A combination of these processes leads to a tree with both tag- and syntactic information allowing for even better analysis, and is used in the Penn and Treebank project [Taylor et al. 2001, p. 7 and 19].

The most common characteristics of such corpora are [Hemming and Lassi 2002, p. 1] [WWW CL2]:

3 SERIAL NLP ARCHITECTURES

It is now time to reflect on the construction of an NLP program, although on a high level. In this chapter, the most basic NLP architecture will be discussed in order to provide an overview of basic concepts and how they are related.

The most basic of the language engineering architectures is the serial architecture. The user feed some sort of data to the system; a piece of text, a scanned paper, an utterance, or any form of natural language input. This triggers a chain of modules to process the data, one after the other, each doing their little part. Due to the inherent nature of a serial architecture, it does not support giving feedback to a preceding module. However, modern variants of the single architecture incorporate this type of feature, resulting in a non-pure serial architecture. Even though this architecture is simple, most traditional NLP systems use it [van Lohuizen 1998, p.3].

Modules, the building-blocks of any NLP architecture, are the focus of the following chapter.

3.1 Modules

The modules that an NLP system consists of can be divided into three types, namely [Knott et al. 2002, p. 64]:

- **Declarative Modules**. This is the name of modules that define declarative (or propositional) knowledge. Declarative knowledge is static facts and beliefs that a person, or in this case NLP system has. Examples of fundamental declarative modules are the corpus-, lexicon- and grammar modules; all of which will be discussed in Chapter 3.2.

- **Procedural Modules**. This is the name of modules that define procedural knowledge. In the real world, this is knowledge about how to perform a certain task. In the NLP context, this refers to the algorithms. Examples of fundamental procedural modules are the Part of Speech tagger and the syntactic parser, of which both are targets for Chapter 3.3.

- **Interface Modules**. These modules enable interaction with the NLP system. These modules have nothing to do with the actual functionality of the system, but instead provide different kinds of users with a possibility to interact with the system. Examples include web based interfaces for normal users, and socket based interfaces for communication with other systems.

To make a comparison with a familiar subject, in the computer world, resources such as memory and hard drives are shared between the running applications. This can be compared to the field of NLP, where declarative modules can be shared between procedural modules in an NLP application. This concept is illustrated in Figure 3.1.

good results agreeing with the teachers 78 % of the time. It should be noted that teachers agree with other teachers 85 % of the time. Critics did not like the idea since it was easy to cheat by for example writing a longer essay, even though adding information might not actually make the essay better. It was also criticized because the analyzing did, for example, not care for important characteristics such as content [Hearst 2000, p. 22].

Today, automatic essay grading systems are delving deeper into the essays, looking at for instance syntactic variety and organization of ideas [Hearst 2000, p. 23]. Some systems have computed grades that have agreed with human grades more than 90 % of the time [Palmer 2002]. However, there are certain aspects of grading longer essays that still are difficult to cope with, for example the rhetoric aspects (the skill in using language effectively and persuasively) [Hearst 2000, p. 34].

Being able to automatically grading essays has several benefits [Valenti 2003, p. 319]:

- **Objective grading**: Grading would be done objectively, instead of subjectively, which can be perceived as unfair by students.

- **Time saving**: Time would be saved since computers can grade essays much faster than their human counter-parts.

If automatic essay grading ever will be able handle all complex aspects of an essay (humoristic and ironic aspects for example), the time (thus money) saved could be used to further increase the gain of knowledge of the students.

2.4 Conclusion

The applications mentioned are not yet perfect, but clearly they all have great potential. For example, being able to produce perfect translations of any media between any two languages will provide all people with more information than ever before in human history.

is to present the user with information gathered from documents suiting a user's request, rather than the documents themselves (which is more common today and called information retrieval, or IR for short), vastly aiding the user in finding relevant information. Due to the complexity of this task, the systems are tailored to a specific domain [Cunningham 1997, p. 3]. For example, a system for extracting weather information from news sources cannot without modification be used to extract information about the bad effects of smoking from medical journals.

According to [Cunningham 1997, p. 4] there are four types of information extraction:

- **Named entity recognition**: This task analyses a text to find names, organizations, dates etc. This system itself is the simplest of the information extraction types but still very powerful. According to The Center for Democracy and Technology, spammers search the web in order to find email addresses from web pages [WWW CDT]. The accuracy at which these entities can be automatically recognized is very high, performing almost the same as humans.

- **Co-reference resolution**: This task finds relationships between entities in the text, which is a needed task for the developers but not for the users. For example, in the sentences "Heard the news about Microsoft? They went bankrupt" the system should be able to understand that "Microsoft" and "They" refer to the same entity. This is a difficult task to automate.

- **Template element construction**: This task associates information that it finds in the text with the entities, for example the location or aliases for a company. Small records of information are the output of this task, like a database record.

- **Scenario template production**: Lastly, scenario template production associates entities with events, such as a company x (the entity) going bankrupt a certain year (the event) etc. This task is not only difficult to automate, but the complexity involved makes it difficult even for a human.

As the information has been gathered, the results can be presented in various forms to suit the user's specific need, and thus vastly simplifying the work needed for a user to do the same task manually.

2.3.3 Automatic Essay Grading

There has been research made in the field of automatic essay grading since the 1960's [Palmer 2002, p. 1222] [Hearst 2000, p. 22]. The meaning of "essay" might be misleading, but in this context it implies both longer essays and free text questions, i.e. "What was the purpose of landing a man on the moon?"

Back in the 1960's, a man by the name of Ellis Page developed an automatic grading system called Project Essay Grader (PEG). The analyzing part of PEG was simple and did not go deep into the actual essay, as it involved such things as the length of the essay, average word length, number of commas, number of prepositions, and so on. Despite the roughness of PEG, the results showed that it actually scored quite

2.3 NLP Applications

This section is dedicated to give practical examples of the usefulness of NLP. The idea is to give some insight into the diversity of NLP applications which exist.

2.3.1 Machine Translation

Using a computer to translate one language to another is called machine translation (MT). [Jurafsky and Martin 2000, p. 799].

One example of a machine translation system freely available to the public is Babel Fish[3]. With this web based system, it is possible to translate complete web pages to and from a number of different languages. These translations are often good enough for a person to understand them, but sometimes they are just gibberish.

IBM is currently working on a Multilingual Automatic Speech-to-Speech Translator (MASTOR) [Liu 2003]. This system would allow for a user to, for example, talk into a device which would translate and synthesize speech in another language. The system is far from perfect however, as it at best translates 51.4 % correctly [Liu 2003, p. 639].

Creating an MT system which can perform well is a complex task, since it is not enough for the system to understand the semantics and pragmatics of the source language; it also has to translate the meaning into another language in a way which people can understand.

For more examples of machine translation implementations see [WWW MT].

2.3.2 Text summarization

Text summarization is the act of transforming a piece of text into a shorter version without modifying its original content. Text summarization can be divided into two variants [Kruengkrai 2003, p. 1]:

- **Information Abstraction**: An information abstraction system deeply analyzes a document or a set of documents using the most advanced natural language processing concepts in order to generate sentences into an abstract of the text. This is more related to natural language understanding, and therefore not further discussed.

- **Information Extraction**: An information extraction system analyzes texts from a specific domain, extracts the most essential sentences, and presents them to the user. This task is simpler than Information Abstraction, as the sentences are extracted as-is, and not generated.

Today, there is an increase in information available to a person, especially due to the World Wide Web [Karkaletsis 1999, p. 123]. The purpose of information extraction

[3] Babel Fish, http://babelfish.altavista.com, last viewed 2004-05-04.

Both sentences contain the word *bear*, but they are not of the same word class. In sentence (1) *bear* is a verb, while in sentence (2) *bear* is a noun.

When giving the computer the task to determine the word class of a specific word, which is a fundamental task in NLP, this information will come in handy.

2.2.2 Syntax

The syntax of a language describes how sentences can be built. The grammar is the set of rules that define the allowed syntax [Grosz 1986, p. 1]. This can be compared to the compile-time error message “Syntax Error” often stumbled upon as a programmer. It means that the code somehow violates the grammar of the programming language. The same rules apply to a natural language. The sentence “I am happy” is syntactically correct, but the sentence “Am happy I” is not - even though they contain the same set of words.

2.2.3 Semantics

Semantics is the study of the meaning of a sentence [Allen 1987, p. 7]. Again, comparing this to programming, this is the same as the logic of the application. Even if the syntax might be correct, you might have logic errors (for-statements indexing arrays past their size, etc). The same is true for natural language. The sentences “Horses have large hoofs” and “Horses have large feet” are both syntactically correct, but the latter makes no sense semantically as horses do not have feet.

2.2.4 Pragmatics

Pragmatics is the study of the use of language in a certain context [Allen 1987, p. 7]. The aim is to find meaning in a sentence looking not only on the sentence itself, but rather on the surrounding environment. Sitting by a table in the next century asking a robot not supporting pragmatics “Can you pass the salt?” would probably result in the answer “Yes” rather than an action - the robot passing the salt. To understand this you can not look at the sentence alone. Another example that is covered by pragmatics is when you refer to objects described previously. For example the sentence “He needs it” is not very informative in itself, since we do not know who “he” is, or what “it” is that he needs.

text. Also, researchers in psychology found that human learning was more statistically-based than they had previously though [Lillian 2003, p. 5].

Today the different methods are used in different places, and it's not uncommon to find algorithms that utilize both approaches. See Chapter 3.3 for more information regarding this.

2.2 The basics of language

Language is a system that allows us to communicate with one another. With the help of *words*, the basic building blocks, sentences are constructed using a set of grammatical rules. These grammatical rules form the allowed *syntax* of the language. The constructed sentence has a certain meaning, which is what you study in *semantics*. The meaning of the sentence might be dependant on the context it is present in, which is the study of *pragmatics* [Allen 1987, p. 7].

This chapter aims to give a brief introduction to these NLP relevant aspects of a language.

2.2.1 Words

Words are the foundation of sentences, so it is logical to examine words before examining sentences. However, as much can be written about words, only word classes will be explained in order to give some insight into the complexity of disambiguating sentences.

Word classes are collections of words that are similar in several ways. Word classes are also known as part-of-speech. Examples of word classes are nouns, verbs and prepositions. When determining a word class, several criteria need to be taken into account [WWW GoE]:

- **The meaning of the word**. One could say that words which are objects, places and people are nouns, and words which are actions are verbs. However, this is a very simplistic way of viewing words, and is not always enough to distinguish word classes. For example, abstract words such as "memory" and "mind" are nouns even though they do not fit the above definition of a noun.

- **The form or shape of the word**. When determining words and their word classes, the form or shape is also considered. For example, words ending with "-tion" are often nouns; for example notion, condition, emotion, commotion, devotion, and so on.

- **The position or environment of the word in a sentence**. Words can look alike but still have different meaning. The meaning of the words is dependent of how they are positioned in the sentence.

  (1) Can you bear with me for just a second?
  (2) I saw a big bear in the woods.

2.1.2 A change of perspective

Not all wanted to believe that language could be analyzed only by looking at statistics. In 1957 the famous linguist Noam Chomsky, inventor of the Context-free Grammar [Jurafsky and Martin 2000], published a paper entitled "Syntactic Structures" explaining why he thought a statistical approach was flawed. He showed what he meant using these two famous sentences [Chomsky 1957, p 3]:

> *(1) Colorless green ideas sleep furiously.*
>
> *(2) Furiously sleep ideas green colorless.*

The first sentence is grammatically correct, but the second is not. He continues by stating [Chomsky 1957, p 4]:

> *It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical model for grammaticalness, these sentences will be ruled out on identical grounds as equally 'remote' from English*

This means that it is not possible to statistically prove the correctness of every sentence, since it is possible to express rare sentences that are not present in the statistical data. A search for sentence (1) and sentence (2) using Google[2] (2004-03-10) yielded 2,550 and 195 results respectively, since Chomsky is quite known in the field of NLP. So ironically this means that, statistically, sentence (1) is much more likely to be correct than sentence (2) (assuming the whole Internet serving as the statistical base).

However, one cannot argue with Chomsky's claim, as there exist many sentences that are grammatically correct, yet no one has written or even said, and therefore statistically, incorrect. This is called the *sparse data problem*. This problem made people dismiss the statistical approaches which had sounded good before and instead look at the *rule-based* (also known as *knowledge-based*) approaches where humans have created rules for the computer to use or where the rules have been automatically deducted [Lillian 2003, p. 4] [Krotov et al. 1999, p. 1].

The sparse data problem might seem to end the idea of using statistical analysis in the field of NLP, but as it would turn out, it did not.

2.1.3 The return of the statistical perspective

In the 1980's, people were beginning yet again to see the statistical approach as a way to help with NLP. The reason for this was that during the 1970's and 1980's this approach was performing much better than the rule-based. In the beginning only speech recognition had benefited from the statistical approach, but it came to engulf all aspects of language analysis. During this time, there were many breakthroughs in machine learning algorithms, enabling computers to learn language from samples of

[2] Google, http://www.google.com, last viewed 2004-05-04

2 NLP OVERVIEW

In this chapter we give an overview of NLP. First a discussion on the history will be given so that the reader can get an understanding of the origins of the major views of NLP problem solving. Then the basic, but still very important elements of language will be briefly described. After that some examples of current NLP applications will be presented, showing that this subject is, and will be of great significance.

2.1 History

It is now time for a brief run-through of NLP history. As is shown, the focus of NLP has changed a great deal during the 20th century. What is important to note is the back and forth between two major views of how to solve problems in NLP: a statistical approach and a rule-based approach.

2.1.1 A statistical perspective

A number of independent sources worked on the field of NLP during the 1940's and 1950's. Zellig Harris believed that much information could be extracted from language data if one knew the relationship, or co-occurrence, between words [Lillian 2003, p. 3]. Fernando Pereira, former researcher at AT&T labs, expresses it a bit humorously [Pereira 2000]:

> *… the co-occurrence of the words "stocks", "bonds" and "bank" in the same passage is potentially indicative of a financial subject matter, and thus tends to disambiguate those word occurrences, reducing the likelihood that the "bank" is a river bank, that the "bonds" are chemical bonds, or that the "stocks" are an ancient punishment device.*

Besides Zellig Harris, Claude Shannon was another pioneer in early statistical approaches to language processing. In 1948 he wrote a paper called "a mathematical theory of communication" that inspired Warren Weaver, who later suggested that the translation between languages should be seen as a cryptographic task, i.e. one language was an encrypted form of another language [Lillian 2003, p. 3].

In 1964 the statisticians Frederick Mosteller and David Wallace published a book called "Inference and Disputed Authorship: The Federalist" explaining how they used statistical methods to resolve the authorship of uncredited federalist papers (since the real author had used a pseudonym instead of his real name). Instead of applying the methods used previously, that is historical data and arguments, they looked at patterns of word occurrences in the papers. The study convinced many people of the true author of the papers [Jurafsky and Martin 2000, p. 12].

So, in the first period of NLP, people viewed language from a *statistical* (also known as *probalistic* or *data-driven*) perspective. However, this was not something that would last.

1.3 Outline

This thesis is roughly divided into two major parts:

- a literature study about core concepts, fundamental algorithms, architectures of significance and other software engineering related topics in the field of NLP;

- an empirical evaluation of an NLP infrastructure called GATE.

In Chapter 2, *NLP Overview*, we begin by giving a brief overview of NLP, including history, basics of language and examples where NLP is applicable. Chapter 3, *Serial NLP Architectures*, is the beginning of more engineering related subjects, such as fundamental NLP concepts and algorithms. Then we continue with the discussion of more advanced architectures in Chapter 4, *Parallel NLP Architectures*. In Chapter 5, *Frameworks and Toolkits*, we present the frameworks and toolkits we found, and select one of them for the experiments. Chapter 6 is solely about the selected framework, which we experiment with in Chapter 7, *An Empirical Evaluation*. Chapter 8 is dedicated to our discussion, and Chapter 9 is the conclusion and future work.

LE is about the construction of applications that involve NLP technology, for instance systems for translating from one language to another, search engines and spelling/grammar checking amongst others. Some examples of where NLP is applicable are given in the NLP overview chapter. The technologies used in LE applications come from different sub-fields of NLP, such as machine translation (MT) and information extraction (IE) [Cunningham 2000, p. 7].

1.1 Problem description

The goal of this thesis is to explore software engineering related questions linked to the development of NLP applications. A big part of this is to isolate the important fundamental concepts and approach these from a software engineering viewpoint. These are the questions intended to illuminate this issue:

- Which architectures are suitable?
- Which are the fundamental concepts?
- Which are the fundamental algorithms?
- Which frameworks and toolkits are available, and are they suitable?
- Which software engineering related problems and questions occur during development?

To elucidate the last question: as mentioned, NLP has more than one side. First there is the linguistic side which we try to avoid as much as possible, and then there is the engineering side which we try to focus on. During the empirical evaluation, we will most likely stumble upon some problems and questions related to software engineering. We will do our best in accumulating these and later take them up for discussion.

1.2 Methodology

In order to find answers to the questions asked, we use two approaches. First we will conduct a literature study. This involves first of all getting a good overview of the subject, in order to be able to select topics that could help in answering the questions. There is a great deal of information available on this subject, and it is difficult to find the gems containing what is considered important information to a software engineer and exclude areas having too much relation to linguistics or NLU.

After the literature study, we will investigate which tools and frameworks are available, if any, for developing NLP systems. Following the investigation, each of these will be examined, one will be selected and experiments will be conducted. The goal is to find relationships between the topics investigated in the literature study and the NLP development environment through experiments. We are in other words approaching the questions from a more concrete perspective.

1 INTRODUCTION

This thesis explores the subject of natural language processing (NLP) from a software engineering perspective.

Even though this thesis is about NLP, it would be a shame to neglect the close relationship to language engineering (LE). There is also a hazy but still relevant connection to natural language understanding (NLU) which needs to be clarified. We will now try to give sufficient definitions of these fields, which is without hesitation a difficult task.

Before we explain these fields, it is relevant to mention that at the bottom of all this lies linguistics, the scientific study of language. Although highly related to NLP, we have tried to stay out of that area, only giving the most basic information needed to understand the rest of the thesis (found in the Overview, Chapter 2).

We have found numerous definitions of NLP, and each and everyone with a disclaimer mentioning that that the field is constantly changing and the definition given does not satisfy everyone. Cunningham expresses it in his eccentric manner [Cunningham 2000, p. 12]:

> *"Attempting to define something as dynamic and multi-faceted as a research field concerned with human language is a difficult task, rather like trying to paint a portrait of a galloping racehorse."*

The definition of NLP that appealed to us the most (relatively new and found in use in multiple places) was one by Elizabeth D. Liddy [Drake 2003, p. 2126]:

> *"Natural language processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications."*

To elucidate, there are multiple techniques to choose from when analyzing texts. The texts should be in a language used by humans (either oral or written). The part "different linguistic levels" alludes to the seven acknowledged levels[1] of analysis at which humans listen, write or in any way process language. NLP is a branch of artificial intelligence, indicated by the part "achieving human-like language processing". In the early days of artificial intelligence, NLP was referred to as NLU. Today, the goal of NLP is in fact NLU even though that state has not yet been reached. NLU tries to achieve human-like language understanding, analyzing texts on a deeper level, being able to draw inferences and answer questions [Drake 2003]. This thesis will try to stay solely on an NLP level, minimizing the effort spent on subjects too much related to the vastly more complicated subject NLU.

[1] The seven levels of linguistic analysis can be read about at the Center of Natural Language Processing (CNLP), http://www.cnlp.org/tech/nlp.asp, last viewed 2004-05-04

CONTENTS

ACKNOWLEDGMENTS

We would like to use this space to thank a small group of entities in a particular order:

- Stefan Johansson, our supervisor;
- Our carbon based family members and girlfriends, for love, support and devotion;
- Kohina, for their miraculously energizing concoction of tone compilations;
- Coke and Coffee, two well known and closely related CNS stimulating mixtures.

Thank you.

ABSTRACT

This thesis is intended to deal with questions related to the processing of naturally occurring texts, also known as natural language processing (NLP). The subject will be approached from a software engineering perspective, and the problem description will be formulated thereafter. The thesis is roughly divided into two major parts. The first part contains a literature study covering fundamental concepts and algorithms. We discuss both serial and parallel architectures, and conclude that different scenarios call for different architectures. The second part is an empirical evaluation of an NLP framework or toolkit chosen amongst a few, conducted in order to elucidate the theoretical part of the thesis. We argue that component based development in a portable language could increase the reusability in the NLP community, where reuse is currently low. The recent emergence of the discovered initiatives and the great potential of many applications in this area reveal a bright future for NLP.

Keywords: Natural Language Processing, Software Engineering, Language Engineering, Architecture.

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 40 weeks of full time studies.

Contact Information:
Authors:
Daniel Åkerud and Henrik Rendlo
E-mail: daniel.akerud@home.se, henrik.rendlo@home.se

University advisor:
Stefan Johansson
Software Engineering and Computer Science

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.tek.bth.se
Phone : +46 457 38 50 00
Fax : +46 457 271 25

Master Thesis
Software Engineering
Thesis no: MSE - 2004:21
June 2004

Natural Language Processing

- from a Software Engineering Perspective

Daniel Åkerud and Henrik Rendlo

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden